
Partially Encrypted Machine Learning using Functional Encryption

Théo Ryffel^{1,2}, Edouard Dufour-Sans¹, Romain Gay^{1,3},
Francis Bach^{2,1} and David Pointcheval^{1,2}

¹Département d'informatique de l'ENS, ENS, CNRS, PSL University, Paris, France

²INRIA, Paris, France

³University of California, Berkeley

{theo.ryffel, edufoursans, romain.gay, francis.bach, david.pointcheval}@ens.fr

Abstract

Machine learning on encrypted data has received a lot of attention thanks to recent breakthroughs in homomorphic encryption and secure multi-party computation. It allows outsourcing computation to untrusted servers without sacrificing privacy of sensitive data. We propose a practical framework to perform partially encrypted and privacy-preserving predictions which combines adversarial training and functional encryption. We first present a new functional encryption scheme to efficiently compute quadratic functions so that the data owner controls what can be computed but is not involved in the calculation: it provides a decryption key which allows one to learn a specific function evaluation of some encrypted data. We then show how to use it in machine learning to partially encrypt neural networks with quadratic activation functions at evaluation time, and we provide a thorough analysis of the information leaks based on indistinguishability of data items of the same label. Last, since most encryption schemes cannot deal with the last thresholding operation used for classification, we propose a training method to prevent selected sensitive features from leaking, which adversarially optimizes the network against an adversary trying to identify these features. This is interesting for several existing works using partially encrypted machine learning as it comes with little reduction on the model's accuracy and significantly improves data privacy.

1 Introduction

As both public opinion and regulators are becoming increasingly aware of issues of data privacy, the area of privacy-preserving machine learning has emerged with the aim of reshaping the way machine learning deals with private data. Breakthroughs in fully homomorphic encryption (FHE) [15, 18] and secure multi-party computation (SMPC) [19, 39] have made computation on encrypted data practical and implementations of neural networks to do encrypted predictions have flourished [34–36, 8, 13].

However, these protocols require the data owner encrypting the inputs and the parties performing the computations to interact and communicate in order to get decrypted results, which we would like to avoid in some cases, like spam filtering, for example, where the email receiver should not need to be online for the email server to classify incoming email as spam or not. Functional encryption (FE) [12, 32] in return does not need interaction to compute over encrypted data: it allows users to receive in plaintext specific functional evaluations of encrypted data: for a function f , a functional decryption key can be generated such that, given any ciphertext with underlying plaintext x , a user can use this key to obtain $f(x)$ without learning x or any other information than $f(x)$. It stands in

between traditional public key encryption, where data can only be directly revealed, and FHE, where data can be manipulated but cannot be revealed: it allows the user to tightly control what is disclosed about his data.

1.1 Use cases

Spam filtering. Consider the following scenario: Alice uses a secure email protocol which makes use of functional encryption. Bob uses Alice’s public key to send her an email, which lands on Alice’s email provider’s server. Alice gave the server keys that enable it to process the email and take a predefined set of appropriate actions without her being online. The server could learn how urgent the email is and decide accordingly whether to alert Alice. It could also detect whether the message is spam and store it in the spam box right away.

Privacy-preserving enforcement of content policies Another use case could be to enable platforms, such as messaging apps, to maintain user privacy through end-to-end encryption, while filtering out content that is illegal or doesn’t adhere to policies the site may have regarding, for instance, abusive speech or explicit images.

These applications are not currently feasible within a reasonable computing time, as the construction of FE for all kinds of circuits is essentially equivalent to *indistinguishable obfuscation* [7, 21], concrete instances of which have been shown insecure, let alone efficient. However, there exist practical FE schemes for the inner-product functionality [1, 2] and more recently for quadratic computations [6], that is usable for practical applications.

1.2 Our contributions

We introduce a new FE scheme to compute quadratic forms which outperforms that of Baltico et al. [6] in terms of complexity, and provide an efficient implementation of this scheme. We show how to use it to build privacy preserving neural networks, which perform well on simple image classification problems. Specifically, we show that the first layers of a polynomial network can be run on encrypted inputs using this quadratic scheme.

In addition, we present an adversarial training technique to process these first layers to improve privacy, so that their output, which is in plaintext, cannot be used by adversaries to recover specific sensitive information at test time. This adversarial procedure is generic for semi-encrypted neural networks and aims at reducing the information leakage, as the decrypted output is not directly the classification result but an intermediate layer (i.e., the neuron outputs of the neural network before thresholding). This has been overlooked in other popular encrypted classification schemes (even in FHE-based constructions like [20] and [15]), where the argmax operation used to select the class label is made in clear, as it is either not possible with FE, or quite inefficient with FHE and SMPC.

We demonstrate the practicality of our approach using a dataset inspired from MNIST [27], which is made of images of digits written using two different fonts. We show how to perform classification of the encrypted digit images in less than 3 seconds with over 97.7% accuracy while making the font prediction a hard task for a whole set of adversaries.

This paper builds on a preliminary version available on the Cryptology ePrint Archive at eprint.iacr.org/2018/206. All code and implementations can be found online at github.com/LaRiffle/collateral-learning and github.com/edufoursans/reading-in-the-dark.

2 Background Knowledge

2.1 Quadratic and Polynomial Neural Networks

Polynomial neural networks are a class of networks which only use linear elements, like fully connected linear layers, convolutions but with average pooling, and model activation functions with polynomial approximations when not simply the square function. Despite these simplifications, they have proved themselves satisfactorily accurate for relatively simple tasks ([20] learns on MNIST and [5] on CIFAR10 [26]). The simplicity of the operations they build on guarantees good efficiency, especially for the gradient computations, and works like [28] have shown that they can achieve convergence rates similar to those of networks with non-linear activations.

In particular, they have been used for several early stage implementations in cryptography [20, 18, 14] to demonstrate the usability of new protocols for machine learning. However, the argmax or other thresholding function present at the end of a classifier network to select the class among the output neurons cannot be conveniently handled, so several protocol implementations (among which ours) run polynomial networks on encrypted inputs, but take the argmax over the decrypted output of the network. This results in potential information leakage which could be maliciously exploited.

2.2 Functional Encryption

Functional encryption extends the notion of public key encryption where one uses a public key pk and a secret key sk to respectively encrypt and decrypt some data. More precisely, pk is still used to encrypt data, but for a given function f , sk can be used to derive a functional decryption key dk_f which will be shared to users so that, given a ciphertext of x , they can decrypt $f(x)$ but not x . In particular, someone having access to dk_f cannot learn anything about x other than $f(x)$. Note also that functions cannot be composed, since the decryption happens within the function evaluation. Hence, only single quadratic functions can be securely evaluated. A formal definition of functional encryption is provided in Appendix A.1.

Perfect correctness. Perfect correctness is achieved in functional encryption: $\forall x \in \mathcal{X}, f \in \mathcal{F}, \Pr[\text{Dec}(\text{dk}_f, \text{ct}) = f(x)] = 1$, where $\text{dk}_f \leftarrow \text{KeyGen}(\text{msk}, f)$ and $\text{ct} \leftarrow \text{Enc}(\text{pk}, x)$. Note that this property is a very strict condition, which is not satisfied by existing fully homomorphic encryption schemes (FHE), such as [16, 22].

2.3 Indistinguishability and security

To assess the security of our framework, we first consider the FE scheme security and make sure that we cannot learn anything more than what the function is supposed to output given an encryption of x . Second, we analyze how sensitive the output $f(x)$ is with respect to the private input x . For both studies, we will rely on *indistinguishability* [23], a classical security notion which can be summed up in the following game: an adversary provides two input items to the challenger (here our FE algorithm), and the challenger chooses one item to be encrypted, runs encryption on it before returning the output. The adversary should not be able to detect which input was used. This is known as IND-CPA security in cryptography and a formal definition of it can be found in Appendix A.2.

We will first prove that our quadratic FE scheme achieves IND-CPA security, then, we will use a relaxed version of indistinguishability to measure the FE output sensitivity. More precisely, we will make the hypothesis that our input data can be used to predict public labels but also sensitive private ones, respectively y_{pub} and y_{priv} . Our quadratic FE scheme q aims at predicting y_{pub} and an adversary would rather like to infer y_{priv} . In this case, the security game consists in the adversary providing two inputs (x_0, x_1) labelled with the same y_{pub} but a different y_{priv} and then trying to distinguish which one was selected by the challenger, given its output $q(x_b)$, $b \in \{0, 1\}$. One way to do this is to measure the ability of an adversary to predict y_{priv} for items which all belong to the same y_{pub} class.

In particular, note that we do not consider approaches based on input reconstruction (as done by [17]) because in many cases, the adversary is not interested in reconstructing the whole input, but rather wants to get insights into specific characteristics.

Another way to see this problem is that we want the sensitive label y_{priv} to be independent from the decrypted output $q(x)$ (which is a proxy to the prediction), given the true public label y_{pub} . This independence notion is known as *separation* and is used as a fairness criterion in [9] if the sensitive features can be misused for discrimination.

3 Our Context for Private Inference

3.1 Classifying in two directions

We are interested in specific types of datasets $(\vec{x}_i)_{i=1, \dots, n}$ which have public labels y_{pub} but also private ones y_{priv} . Moreover, these different types of labels should be entangled, meaning that they should not be easily separable, unlike the color and the shape of an object in an image for example which can be simply separated. For example, in the spam filtering use case mentioned above, y_{pub} would be a spam flag, and y_{priv} would be some marketing information highlighting areas of interest

of the email recipient like technology, culture, etc. In addition, to simplify our analysis, we assume that classes are balanced for all types of labels, and that labels are independent from each other given the input: $\forall \vec{x}, P(y_{\text{pub}}, y_{\text{priv}} | \vec{x}) = P(y_{\text{pub}} | \vec{x}) P(y_{\text{priv}} | \vec{x})$. To illustrate our approach in the case of image recognition, we propose a synthetic dataset inspired by MNIST which consists of 60 000 grey scaled images of 28×28 pixels representing digits using two fonts and some distortion, as shown in Figure 1. Here, the public label y_{pub} is the digit on the image and the private one y_{priv} is the font used to draw it.



Figure 1: Artificial dataset inspired from MNIST with two types of labels.

We define two tasks: a *main* task which tries to predict y_{pub} using a partially-encrypted polynomial neural network with functional encryption, and a *collateral* task which is performed by an adversary who tries to leverage the output of the FE encrypted network at test time to predict y_{priv} . Our goal is to perform the main task with high accuracy while making the collateral one as bad as random predictions. In terms of indistinguishability, given a dataset with the same digit drawn, it should be infeasible to detect the used font.

3.2 Equivalence with a Quadratic Functional Encryption scheme

We now introduce our new framework for quadratic functional encryption and show that it can be used to partially encrypt a polynomial network.

3.2.1 Functional Encryption for Quadratic Polynomials

We build an efficient FE scheme for the set of quadratic functions defined as $\mathcal{F}_{n, B_x, B_y, B_q} \subset \{q : [-B_x, B_x]^n \times [-B_y, B_y]^n \rightarrow \mathbb{Z}\}$, where q is described as a set of bounded coefficients $\{q_{i,j} \in [-B_q, B_q]\}_{i,j \in [n]}$ and for all vectors (\vec{x}, \vec{y}) , we have $q(\vec{x}, \vec{y}) = \sum_{i,j \in [n]} q_{i,j} x_i y_j$.

A complete version of our scheme is given in Figure 2, but here are the main ideas and notations. First note that we use bilinear groups, *i.e.*, a set of prime-order groups $\mathbb{G}_1, \mathbb{G}_2$ and \mathbb{G}_T together with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ called *pairing* which satisfies $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for any exponents $a, b \in \mathbb{Z}$: one can compute quadratic polynomials in the exponent. Here, g_1, g_2 are generators of \mathbb{G}_1 and \mathbb{G}_2 and $g_T := e(g_1, g_2)$ is a generator of the target group \mathbb{G}_T . A pair of vectors (\vec{s}, \vec{t}) is first selected and constitutes the private key msk , while the public key is $(g_1^{\vec{s}}, g_2^{\vec{t}})$.

Encrypting (\vec{x}, \vec{y}) roughly consists of masking $g_1^{\vec{x}}$ and $g_2^{\vec{y}}$ with $g_1^{\vec{s}}$ and $g_2^{\vec{t}}$, which allows any user to compute $g_T^{q(\vec{x}, \vec{y}) - q(\vec{s}, \vec{t})}$ with for any quadratic function q , using the pairing. The functional decryption key for a specific q is $g_T^{q(\vec{s}, \vec{t})}$ which allows to get $g_T^{q(\vec{x}, \vec{y})}$. Last, taking the discrete logarithm gives access to $q(\vec{x}, \vec{y})$ (discrete logarithm for small exponents is easy). Security uses the fact that it is hard to compute msk from pk (discrete logarithm for large exponents \vec{s}, \vec{t} is hard to compute). More details are given in Appendix B.¹

Theorem 3.1 (Security, correctness and complexity) *The FE scheme provided in Figure 2:*

- is IND-CPA secure in the Generic Bilinear Group Model,
- verifies $\log(\text{out}) = q(\vec{x}, \vec{y})$ and satisfies perfect correctness,
- has a overall decryption complexity of $2n^2(E + P) + P + D$,

where E, P and D respectively denote exponentiation, pairing and discrete logarithm complexities.

Our scheme outperforms previous schemes for quadratic FE with the same security assumption, like the one from [6, Sec. 4] which achieves $3n^2(E + P) + 2P + D$ complexity and uses larger ciphertexts and decryption keys. Note that the efficiency of the decryption can even be further optimized for those quadratic polynomials used that are relevant to our application (see Section 3.2.2).

¹Note that we only present a simplified scheme here. In particular, the actual encryption is randomized, which is necessary to achieve IND-CPA security.

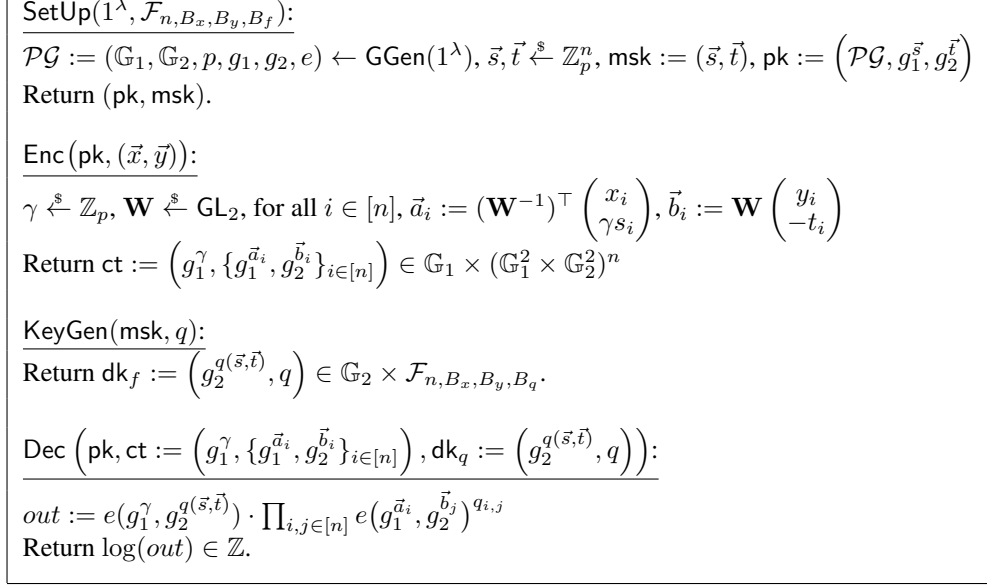


Figure 2: Our functional encryption scheme for quadratic polynomials.

Computing the discrete logarithm for decryption. Our decryption requires computing discrete logarithms of group elements in base g_T , but contrary to previous works like [25] it is independent of the ciphertext and the functional decryption key used to decrypt. This allows to pre-compute values and dramatically speeds-up decryption.

3.2.2 Equivalence of the FE scheme with a Quadratic Network

We classify data which can be represented as a vector $\vec{x} \in [0, B]^n$ (in our case, the size $B = 255$, and the dimension $n = 784$) and we first build models $(q_i)_{i \in [\ell]}$ for each public label $i \in [\ell]$, such that our prediction y_{pub} for \vec{x} is $\text{argmax}_{i \in [\ell]} q_i(\vec{x})$.

Quadratic polynomial on \mathbb{R}^n . The most straightforward way to use our FE scheme would be for us to learn a model $(\mathbf{Q}_i)_{i \in [\ell]} \in (\mathbb{R}^{n \times n})^\ell$, which we would then round onto integers, such that $q_i(\vec{x}) = \vec{x}^\top \mathbf{Q}_i \vec{x}, \forall i \in [\ell]$. This is a unnecessarily powerful model in the case of MNIST as it has ℓn^2 parameters ($n = 784$), and the resulting number of pairings to compute would be unreasonably large.

Linear homomorphism. The encryption algorithm of our FE scheme is linearly homomorphic with respect to the plaintext: given an encryption of (\vec{x}, \vec{y}) under the secret key $\text{msk} := (\vec{s}, \vec{t})$, one can efficiently compute an encryption of $(\vec{u}^\top \vec{x}, \vec{v}^\top \vec{y})$ under the secret key $\text{msk}' := (\vec{u}^\top \vec{s}, \vec{v}^\top \vec{t})$ for any linear combination \vec{u}, \vec{v} (see proof in Appendix B.2). Any vector \vec{v} is a column, and \vec{v}^\top is a row.

Therefore, if q can be written $q(\vec{x}, \vec{y}) = (\mathbf{U}\vec{x})^\top \mathbf{M}(\mathbf{V}\vec{y})$ for all (\vec{x}, \vec{y}) , with $\mathbf{U}, \mathbf{V} \in \mathbb{Z}_p^{d \times n}$ projection matrices and $\mathbf{M} \in \mathbb{Z}_p^{d \times d}$, it is more efficient to first compute the encryption of $(\mathbf{U}\vec{x}, \mathbf{V}\vec{y})$ from the encryption of (\vec{x}, \vec{y}) , and then to apply the functional decryption on these ciphertexts, because their underlying plaintexts are of reduced dimension $d < n$. This reduces the number of exponentiations from $2n^2$ to $2dn$ and the number of pairing computations from $2n^2$ to $2d^2$ for a single q_i . This is a major efficiency improvement for small d , as pairings are the main bottleneck in the computation.

Projection and quadratic polynomial on \mathbb{R}^d . We can use this and apply the quadratic polynomials on projected vectors: we learn $\mathbf{P} \in \mathbb{R}^{n \times d}$ and $(\mathbf{Q}_i)_{i \in [\ell]} \in (\mathbb{R}^{d \times d})^\ell$, and our model is $q_i(\vec{x}) = (\mathbf{P}\vec{x})^\top \mathbf{Q}_i(\mathbf{P}\vec{x}), \forall i \in [\ell]$. We only need $2\ell d^2$ pairings and since the same \mathbf{P} is used for all q_i , we only compute once the encryption of $\mathbf{P}\vec{x}$ from the encryption of \vec{x} . Better yet, we can also perform the pairings only once, and then compute the scores by exponentiating with different coefficients the same results of the pairings, thus only requiring $2d^2$ pairing evaluations, independently of ℓ .

Degree 2 polynomial network, with one hidden layer. To further reduce the number of pairings, we actually limit ourselves to diagonal matrices, and thus rename \mathbf{Q}_i to \mathbf{D}_i . We find that the gain in efficiency associated with only computing $2d$ pairings is worth the small drop in accuracy. The resulting model is actually a polynomial network of degree 2 with one hidden layer of d neurons and the activation function is the square. In the following experiments we take $d = 40$.

Our final encrypted model can thus be written as $q_i(\vec{x}) = (\mathbf{P}\vec{x})^\top \mathbf{D}_i(\mathbf{P}\vec{x}), \forall i \in [\ell]$, where we add a bias term to \vec{x} by replacing it with $\vec{x} = (1 \ x_1 \dots x_n)$.

Full network. The result of the quadratic $(q_i(\vec{x}))_{i \in [\ell]}$ (i.e., of the private quadratic network) is now visible in clear. As mentioned above, we cannot compose this block several times as it contains decryption, so this is currently the best that we can have as an encrypted computation with FE. Instead of simply applying the argmax to the cleartext output of this privately-evaluated quadratic network to get the label, we observe that adding more plaintext layers on top of it helps improving the overall accuracy of the main task. We have therefore a neural network composed of a private and a public part, as illustrated in Figure 3.

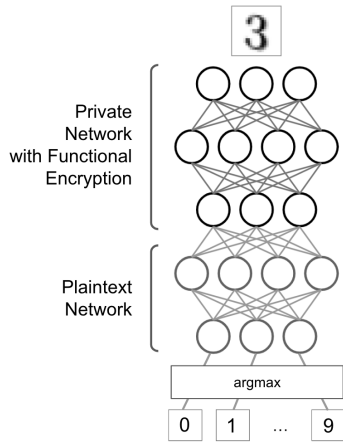


Figure 3: Semi-encrypted network using quadratic FE.

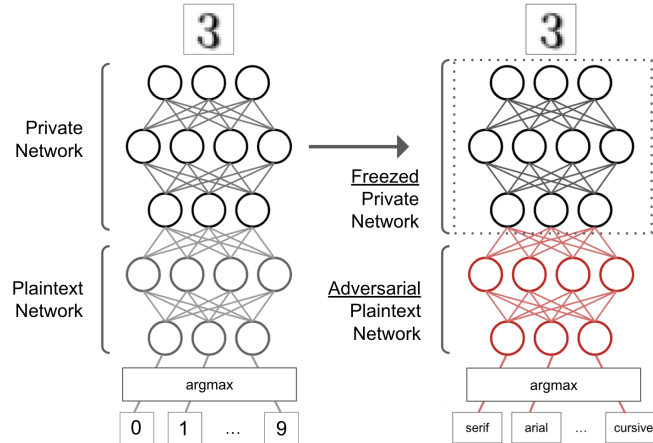


Figure 4: Semi-encrypted network with an adversary trying to recover private labels from the private quadratic network.

3.3 Threat of Collateral Learning

A typical adversary would have a read access to the main task classification process. It would leverage the output of the quadratic network to try to learn the font used on ciphered images. To do this, all that is needed is to train another network on top of the quadratic network so that it learns to predict the font, assuming some access to labeled samples (which is the case if the adversary encrypts itself images and provides them to the main task at evaluation time). Note that in this case the private network is not updated by the collateral network as we assume it is only provided in read access after the main task is trained. Figure 4 summarizes the setting.

We implemented this scenario using as adversary a neural network composed of a first layer acting as a decompression step where we increase the number of neurons from 10 back to 28×28 and add on top of it a classical² convolutional neural network (CNN). This structure is reminiscent of autoencoders [38] where the bottleneck is the public output of the private net and the challenge of this autoencoder is to correctly memorize the public label while forgetting the private one. What we observed is striking: in less than 10 epochs, the collateral network leverages the 10 public neurons output and achieves 93.5% accuracy for the font prediction. As expected, it gets even worse when the adversary is assessed with the indistinguishability criterion because in that case the adversary can work on a dataset where only a specific digit is represented: this reduces the variability of the samples and makes it easier to distinguish the font; the probability of success is indeed of 96.9%.

We call *collateral learning* this phenomenon of learning unexpected features and will show in the next section how to implement counter-measures to this threat in order to improve privacy.

²<https://github.com/pytorch/examples/blob/master/mnist/main.py>

4 Defeating Collateral Learning

4.1 Reducing information leakage

Our first approach is based on the observation that we leak many bits of information. We first investigate whether we can reduce the number of outputs of the privately-evaluated network, as adding extra layers on top of the private network makes it no longer necessary to keep 10 of them. The intuition is that if the information that is relevant to the main task can fit in less than 10 neurons, then the extra neurons would leak unnecessary information. We have therefore a trade-off between reducing too much and losing desired information or keeping a too large output and having an important leakage. We can observe this through the respective accuracies as it is shown in Figure 5, where the main and adversarial networks are CNNs as in Section 3.3 with 10 epochs of training using 7-fold cross validation. What we observe here is interesting: the main task does not exhibit significant weaknesses even with size 3 where we drop to 97.1% which is still very good although 2% under the best accuracy. In return, the collateral accuracy starts to significantly decrease when output size is below 7. At size 4, it is only 76.4% on average so 18% less than the baseline. We will keep an output size of 3 or 4 for the next experiments to keep the main accuracy almost unchanged.

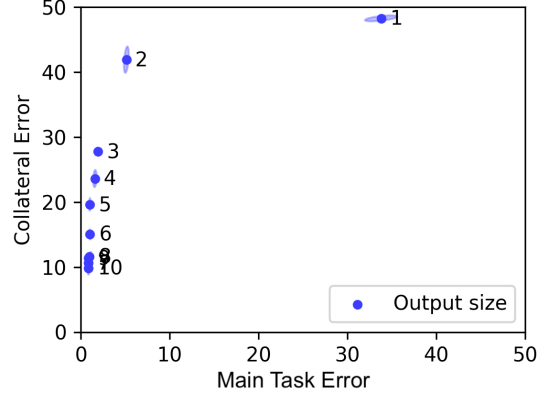


Figure 5: Trade-off between main and collateral accuracies depending on the private output size.

Another hyperparameter that we can consider is the weight compression: how many bits do we need to represent the weights on the private networks layers? This is of interest for the FE scheme as we need to convert all weights to integers and those integers will be low provided that the compression rate is high. Small weight integers mean that the output of the private network has a relatively low amplitude and can be therefore efficiently decrypted using discrete logarithm. We managed to express all weights and even the input image using 4 bit values with limited impact on the main accuracy and almost none on the collateral one. Details about compression can be found in Appendix C.1.

4.2 Adversarial training

We propose a new approach to actively adapt against collateral learning. The main idea is to simulate adversaries and to try to defeat them. To do this, we use semi-adversarial training and optimize simultaneously the main classification objective and the opposite of the collateral objective of a given simulated adversary. The function that we want to minimize at each iteration step can be written:

$$\min_{\theta_q} [\min_{\theta_{\text{pub}}} \mathcal{L}_{\text{pub}}(\theta_q, \theta_{\text{pub}}) - \alpha \min_{\theta_{\text{pub}}} \mathcal{L}_{\text{priv}}(\theta_q, \theta_{\text{pub}})].$$

This approach is inspired from [29] where the authors train some objective against nuisances parameters to build a classifier independent from these nuisances. Private features leaking in our scheme can indeed be considered to be a nuisance. However, our approach goes one step further as we do not just stack a network above another; our global network structure is fork-like: the common basis is the private network and the two forks are the main and collateral classifiers. This allows us to have a better classifier for the main task which is not as sensitive to the adversarial training as the scheme exposed by [29, Figure 1]. One other difference is that the collateral classifier is a specific modeling of an adversary, and we will discuss this in details in the next section. We define in Figure 6 the 3-step procedure used to implement this semi-adversarial training using partial back-propagation.

5 Experimental Results

Accurate main task and poor collateral results. In Figures 7 and 8 we show that the output size has an important influence on the two tasks' performances. For this experiment, we use $\alpha = 1.7$ as detailed in Appendix C.2, the adversary uses the same CNN as stated above and the main network is

<p>Pre-training: <i>Initial phase where both tasks learn and strengthen before the joint optimization</i></p> <p>Minimize $\mathcal{L}_{\text{pub}}(\theta_q, \theta_{\text{pub}})$</p> <p>Minimize $\mathcal{L}_{\text{priv}}(\text{Frozen}(\theta_q), \theta_{\text{priv}})$</p>
<p>Semi-adversarial training: <i>The joint optimization phase, where θ_{pub} and θ_{priv} are updated depending on the variations of θ_q and θ_q is optimized to reduce the loss $\mathcal{L} = \mathcal{L}_{\text{pub}} - \alpha\mathcal{L}_{\text{priv}}$</i></p> <p>Minimize $\mathcal{L}_{\text{pub}}(\text{Frozen}(\theta_q), \theta_{\text{pub}})$</p> <p>Minimize $\mathcal{L}_{\text{priv}}(\text{Frozen}(\theta_q), \theta_{\text{priv}})$</p> <p>Minimize $\mathcal{L} = \mathcal{L}_{\text{pub}}(\theta_q, \text{Frozen}(\theta_{\text{pub}})) - \alpha\mathcal{L}_{\text{priv}}(\theta_q, \text{Frozen}(\theta_{\text{pub}}))$</p>
<p>Recover phase: <i>Both tasks recover from the perturbations induced by the adversarial phase, θ_q does not change anymore</i></p> <p>Minimize $\mathcal{L}_{\text{pub}}(\text{Frozen}(\theta_q), \theta_{\text{pub}})$</p> <p>Minimize $\mathcal{L}_{\text{priv}}(\text{Frozen}(\theta_q), \theta_{\text{priv}})$</p>

Figure 6: Our semi-adversarial training scheme.

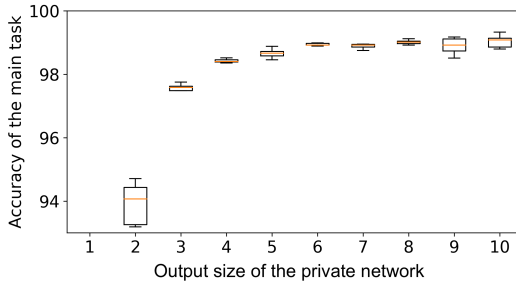


Figure 7: Influence of the output size on the main task accuracy with adversarial training.

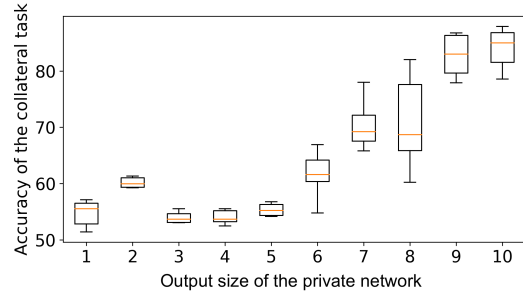


Figure 8: Influence of the output size on the collateral task accuracy with adversarial training.

a simple feed forward network (FFN) with 4 layers. We observe that both networks behave better when the output size increases, but the improvement is not synchronous which makes it possible to have a main task with high accuracy while the collateral task is still very inaccurate. In our example, this corresponds to an output size between 3 and 5. Note that the collateral result is the accuracy at the distinction task, i.e., the digit is fixed for the adversary which trains to distinguish two fonts during a 50 epoch *recover phase* using 7-fold cross validation, after 50 epochs of *semi-adversarial training* have been spent to reduce leakage from the private network.

Generalizing resistance against multiple adversaries. In practice, it is very likely that the adversary will use a different model than the one against which the protection has been built. We have therefore investigated how building resistance against a model M can provide resistance against other models. Our empirical results tend to show that models with less parameters than M do not perform well. In return, models with more parameters can behave better, provided that the complexity does not get excessive for the considered task, because it would not provide any additional advantage and would just lead to learning noise. In particular, the CNN already mentioned above seems to be a sufficiently complex model to resist against a wide range of feed forward (FFN) and convolutional networks, as illustrated in Figure 9 where the measure used is indistinguishability of the font for a fixed digit. This study is not exhaustive as the adversary can change the activation function (here we use relu) or even the training parameters (optimizer, batch size, dropout, etc.), but these do not seem to provide any decisive advantage.

We also assessed the resistance to a large range of other models from the sklearn library [33] and report the collateral accuracy in Figure 10. As can be observed, some models such as k-nearest neighbors or random forests perform better compared to neural networks, even if their accuracy remains relatively low. One reason can be that they operate in a very different manner compared to the model on which the adversarial training is performed: k-nearest neighbors for example just considers distances between points.

Runtime. Training in semi-adversarial mode can take quite a long time depending on the level of privacy one wants to achieve. However, the runtime during the test phase is much faster, it is dominated by the FE scheme part which can be broken down to 4 steps: functional key generation,

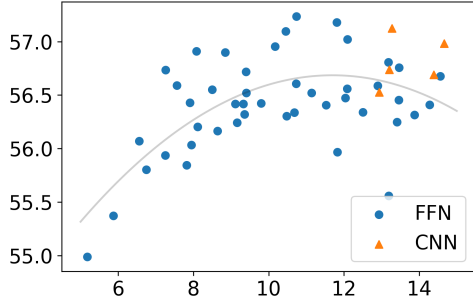


Figure 9: Collateral accuracy depending of the adversarial network complexity seen as the log of the number of parameters.

Linear Ridge Regression	$53.5 \pm 0.5\%$
Logistic Regression	$52.5 \pm 0.6\%$
Quad. Discriminant Analysis	$54.9 \pm 0.3\%$
SVM (RBF kernel)	$57.9 \pm 0.4\%$
Gaussian Process Classifier	$53.8 \pm 0.3\%$
Gaussian Naive Bayes	$53.2 \pm 0.5\%$
K-Neighbors Classifier	$58.1 \pm 0.7\%$
Decision Tree Classifier	$56.8 \pm 0.4\%$
Random Forest Classifier	$58.9 \pm 0.2\%$
Gradient Boosting Classifier	$58.9 \pm 0.2\%$

Figure 10: Accuracy on the distinction task for different adversarial learning models.

encryption of the input, evaluation of the function and discrete logarithm. Regarding encryption and evaluation, the main overhead comes from the exponentiations and pairings which are implemented in the crypto library charm [3]. In return, the discrete logarithm is very efficient thanks to the reduction of the weights amplitude detailed in Figure 4.1.

Functional key generation	$94 \pm 5\text{ms}$	Evaluation time	$2.97 \pm 0.07\text{s}$
Encryption time	$12.1 \pm 0.3\text{ s}$	Discrete logarithms time	$24 \pm 9\text{ms}$

Table 1: Average runtime for the FE scheme using a 2,7 GHz Intel Core i7 and 16GB of RAM.

Table 1 shows that encryption time is longer than evaluation time, but a single encryption can be used with several decryption keys dk_{q_i} to perform multiple evaluation tasks.

6 Conclusion

We have shown that functional encryption can be used for practical applications where machine learning is used on sensitive data. We have raised awareness about the potential information leakage when not all the network is encrypted and have proposed semi-adversarial training as a solution to prevent targeted sensitive features from leaking for a vast family of adversaries.

However, it remains an open problem to provide privacy-preserving methods for all features except the public ones as they can be hard to identify in advance. On the cryptography side, extension of the range of functions supported in functional encryption would help increase provable data privacy, and adding the ability to hide the function evaluated would be of interest for sensitive neural networks.

Acknowledgments

This work was supported in part by the European Community’s Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud), the European Community’s Horizon 2020 Project FENTEC (Grant Agreement no. 780108), the Google PhD fellowship, and the French FUI ANBLIC Project.

References

- [1] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, Heidelberg, March / April 2015.
- [2] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, August 2016.

- [3] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.
- [4] Miguel Ambrona, Gilles Barthe, Romain Gay, and Hoeteck Wee. Attribute-based encryption in the generic group model: Automated proofs and new constructions. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 647–664. ACM Press, October / November 2017.
- [5] Ahmad Al Badawi, Jin Chao, Jie Lin, Chan Fook Mun, Jun Jie Sim, Benjamin Hong Meng Tan, Xiao Nan, Khin Mi Mi Aung, and Vijay Ramaseshan Chandrasekhar. The alexnet moment for homomorphic encryption: Hcnn, the first homomorphic cnn on encrypted data with gpus. Cryptology ePrint Archive, Report 2018/1056, 2018.
- [6] Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 67–98. Springer, Heidelberg, August 2017.
- [7] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.
- [8] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Privacy-preserving ecg classification with branching programs and neural networks. *IEEE Transactions on Information Forensics and Security*, 6(2):452–468, 2011.
- [9] Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and Machine Learning*. fairml-book.org, 2018.
- [10] Gilles Barthe, Edvard Fagerholm, Dario Fiore, John C. Mitchell, Andre Scedrov, and Benedikt Schmidt. Automated analysis of cryptographic assumptions in generic group models. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 95–112. Springer, Heidelberg, August 2014.
- [11] Dan Boneh and Matthew K. Franklin. Identity based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.
- [12] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.
- [13] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, volume 4324, page 4325, 2015.
- [14] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. Cryptology ePrint Archive, Report 2017/1114, 2017. <https://eprint.iacr.org/2017/1114>.
- [15] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, pages 483–512, 2018.
- [16] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Oct 2011.
- [17] Sergiu Carpov, Caroline Fontaine, Damien Ligier, and Renaud Sirdey. Illuminating the dark or how to recover what should not be seen. *IACR Cryptology ePrint Archive*, 2018:1001, 2018.
- [18] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016*, pages 3–33, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

- [19] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 643–662, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- [20] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. Technical report, February 2016.
- [21] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.
- [22] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.
- [23] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.
- [24] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, September 2004.
- [25] Sam Kim, Kevin Lewi, Avradip Mandal, Hart Montgomery, Arnab Roy, and David J. Wu. Function-hiding inner product encryption is practical. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 544–562. Springer, Heidelberg, September 2018.
- [26] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.
- [27] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.
- [28] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. *CoRR*, abs/1410.1141, 2014.
- [29] Gilles Louppe, Michael Kagan, and Kyle Cranmer. Learning to pivot with adversarial networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 981–990. Curran Associates, Inc., 2017.
- [30] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005.
- [31] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.
- [32] Adam O’Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.
- [33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [34] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, ASIACCS ’18*, pages 707–721, New York, NY, USA, 2018. ACM.

- [35] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *CoRR*, abs/1811.04017, 2018.
- [36] Microsoft SEAL (release 3.2). <https://github.com/Microsoft/SEAL>, February 2019. Microsoft Research, Redmond, WA.
- [37] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.
- [38] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.
- [39] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: Efficient and private neural network training. (PETS 2019), February 2019.

A Functional Encryption and crypto tools

A.1 Formal definition of Functional Encryption

Functional encryption relies on a pair of keys like in public key encryption: a master secret key msk and a public key pk . The public key pk can be shared and is used to encrypt the data, while the master secret key msk is used to build functional decryption keys dk_f for $f \in \mathcal{F}$. A user having access to c an encryption of x with pk and to dk_f can learn $f(x)$ but can't learn anything else about x .

We give the definition of Functional Encryption, originally defined in [12, 32].

Definition A.1 (Functional Encryption) A functional encryption scheme FE for a set of functions $\mathcal{F} \subseteq \mathcal{X} \rightarrow \mathcal{Y}$ is a tuple of PPT algorithms $\text{FE} = (\text{SetUp}, \text{KeyGen}, \text{Enc}, \text{Dec})$ defined as follows.

$\text{SetUp}(1^\lambda, \mathcal{F})$ takes as input a security parameter 1^λ , the set of functions \mathcal{F} , and outputs a master secret key msk and a public key pk .

$\text{KeyGen}(\text{msk}, f)$ takes as input the master secret key and a function $f \in \mathcal{F}$, and outputs a functional decryption key dk_f .

$\text{Enc}(\text{pk}, x)$ takes as input the public key pk and a message $x \in \mathcal{X}$, and outputs a ciphertext ct .

$\text{Dec}(\text{dk}_f, \text{ct})$ takes as input a functional decryption key dk_f and a ciphertext ct , and returns an output $y \in \mathcal{Y} \cup \{\perp\}$, where \perp is a special rejection symbol.

A.2 IND-CPA security

With notations of Appendix A.1, for any stateful adversary \mathcal{A} and any functional encryption scheme FE , we define the following advantage.

$$\text{Adv}_{\mathcal{A}}^{\text{FE}}(\lambda) := \Pr \left[\begin{array}{l} (\text{pk}, \text{msk}) \leftarrow \text{SetUp}(1^\lambda, \mathcal{F}) \\ (x_0, x_1) \leftarrow \mathcal{A}^{\text{KeyGen}(\text{msk}, \cdot)}(\text{pk}) \\ \beta \xleftarrow{\$} \{0, 1\}, \text{ct} \leftarrow \text{Enc}(\text{pk}, x_\beta) \\ \beta' \leftarrow \mathcal{A}^{\text{KeyGen}(\text{msk}, \cdot)}(\text{ct}) \end{array} \right] - \frac{1}{2},$$

with the restriction that all queries f that \mathcal{A} makes to key generation algorithm $\text{KeyGen}(\text{msk}, \cdot)$ must satisfy $f(x_0) = f(x_1)$.

We say FE is IND-CPA secure if for all PPT adversaries \mathcal{A} , $\text{Adv}_{\mathcal{A}}^{\text{FE}}(\lambda) = \text{negl}(\lambda)^3$.

A.3 Bilinear Groups

Our FE scheme uses bilinear (or *pairing*) groups, whose use in cryptography has been introduced by [11, 24]. More precisely, given λ a security parameter, let \mathbb{G}_1 and \mathbb{G}_2 be two cyclic groups of prime order p (for a 2λ -bit prime p) and g_1 and g_2 their generators, respectively. The application $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a pairing if it is efficiently computable, non-degenerated, and bilinear: $e(g_1^\alpha, g_2^\beta) = e(g_1, g_2)^{\alpha\beta}$ for any $\alpha, \beta \in \mathbb{Z}_p$. Additionally, we define $g_T := e(g_1, g_2)$ which spans the group \mathbb{G}_T of prime order p .

We will denote by GGen a probabilistic polynomial-time (PPT) algorithm that on input 1^λ returns a description $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, p, g_1, g_2, e)$ of an asymmetric bilinear group. For convenience, given $s = 1, 2$ or T , $n \in \mathbb{N}$ and vectors $\vec{u} := (u_1 \dots u_n) \in \mathbb{Z}_p^n$, $\vec{v} \in \mathbb{Z}_p^n$, we denote by $g_s^{\vec{u}} := (g_s^{u_1} \dots g_s^{u_n}) \in \mathbb{G}_s^n$ and $e(g_1^{\vec{u}}, g_2^{\vec{v}}) = \prod_{i=1}^n e(g_1, g_2)^{u_i v_i} = e(g_1, g_2)^{\vec{u} \cdot \vec{v}} \in \mathbb{G}_T$, where $\vec{u} \cdot \vec{v}$ is the inner product, i.e. $\vec{u} \cdot \vec{v} := \sum_{i=1}^n u_i v_i$.

³In cryptography, the security parameter λ is a measure of the probability with which an adversary can break the scheme. λ or 1^λ means that the probability of breaking the scheme is $2^{-\lambda}$.

B Our Quadratic Functional Encryption Scheme

B.1 Proofs of IND-CPA security and correctness

Proof of Security

To prove security of our scheme, we use the Generic Bilinear Group Model, which captures the fact that no attacks can make use of the representation of group elements. For convenience, we use Maurer’s model [30], where a third party implements the group and gives access to the adversary via handles, providing also equality checking. This is an alternative, but equivalent, formulation of the Generic Group Model, as originally introduced in [31, 37].

We prove security in two steps: first, we use a master theorem from [6] that relates the security in the Generic Bilinear Group model to a security in a symbolic model. Second, we prove security in the symbolic model. Let us now explain the symbolic model (the next paragraph is taken verbatim from [4]).

In the symbolic model, the third party does not implement an actual group, but keeps track of abstract expressions. For example, consider an experiment where values x, y are sampled from \mathbb{Z}_p and the adversary gets handles to g^x and g^y . In the generic model, the third party will choose a group of order p , for example $(\mathbb{Z}_p, +)$, will sample values $x, y \leftarrow_R \mathbb{Z}_p$ and will give handles to x and y . On the other hand, in the symbolic model the sampling won’t be performed and the third party will output handles to X and Y , where X and Y are abstract variables. Now, if the adversary asks for equality of the elements associated to the two handles, the answer will be negative in the symbolic model, since abstract variable X is different from abstract variable Y , but there is a small chance the equality check succeeds in the generic model (only when the sampling of x and y coincides).

To apply the master theorem, we first need to change the distribution of the security game to ensure that the public key, challenge ciphertext, and functional decryption keys only contain group elements whose exponent is a polynomial evaluated on uniformly random values in \mathbb{Z}_p (this is called polynomially induced distributions in [6, Definition 10], and previously in [10]). We show that this is possible with only a negligible statistical change in the distribution of the adversary view.

After applying the master theorem from [6], we prove the security in the symbolic model (cf. Appendix D.1), which simply consists of checking that an algebraic condition on the scheme is satisfied.

Theorem B.1 (IND-CPA Security in the Generic Bilinear Group Model) *For any PPT adversary \mathcal{A} that performs at most Q group operations against the functional encryption scheme described on 2, we have, in the generic bilinear group model:*

$$\text{Adv}_{\mathcal{A}}^{\text{FE}}(\lambda) \leq \frac{12 \cdot (6n + 3 + Q + Q')^2 + 1}{p},$$

where Q' is the number of queries to $\text{KeyGen}(\text{msk}, \cdot)$.

The proof of this result is quite technical and can be found in the dedicated Appendix D.

Proof of Correctness

For all $i, j \in [n]$, we have:

$$e(g_1^{\vec{a}_i}, g_2^{\vec{b}_j}) = g_T^{\vec{a}_i \cdot \vec{b}_j} = g_T^{x_i y_j - \gamma s_i t_j}$$

since

$$\begin{aligned} \vec{a}_i \cdot \vec{b}_j &= \left((\mathbf{W}^{-1})^\top \begin{pmatrix} x_i \\ \gamma s_i \end{pmatrix} \right)^\top \cdot \left(\mathbf{W} \begin{pmatrix} y_j \\ -t_j \end{pmatrix} \right) \\ &= \begin{pmatrix} x_i \\ \gamma s_i \end{pmatrix}^\top \mathbf{W}^{-1} \mathbf{W} \begin{pmatrix} y_j \\ -t_j \end{pmatrix} = x_i y_j - \gamma s_i t_j. \end{aligned}$$

Therefore we have:

$$\begin{aligned} out &= e(g_1^\gamma, g_2^{q(\vec{s}, \vec{t})}) \cdot \prod_{i,j} e(g_1^{\vec{a}_i}, g_2^{\vec{b}_j})^{q_{i,j}} = g_T^{\gamma q(\vec{s}, \vec{t})} \cdot g_T^{\sum_{i,j} q_{i,j} x_i y_j - \gamma q_{i,j} s_i t_j} \\ &= g_T^{\gamma q(\vec{s}, \vec{t})} \cdot g_T^{q(\vec{x}, \vec{y}) - \gamma q(\vec{s}, \vec{t})} = g_T^{q(\vec{x}, \vec{y})}. \end{aligned}$$

Proof of Complexity

The complexity can be inferred from the decryption phase as detailed in Figure 2 and we compare this with previous quadratic FE schemes in Figure 11.

FE scheme	ct	dk_f	Dec	Assumption
[6, Sec. 3]	$\mathbb{G}_1^{6n+1} \times \mathbb{G}_2^{6n+1}$	$\mathbb{G}_1 \times \mathbb{G}_2$	$6n^2(E_1 + P) + 2P$	SXDH, 3PDDH
[6, Sec. 4]	$\mathbb{G}_1^{2n+1} \times \mathbb{G}_2^{2n+1}$	\mathbb{G}_1^2	$3n^2(E_1 + P) + 2P$	GGM
Ours	$\mathbb{G}_1^{2n+1} \times \mathbb{G}_2^{2n}$	\mathbb{G}_2	$2n^2(E_1 + P) + P$	GGM

Figure 11: Performance comparison of FE for quadratic polynomials. E_1 and P denote exponentiation in \mathbb{G}_1 and pairing evaluation, respectively. Decryption additionally requires solving a discrete logarithm but this computational overhead is the same for all schemes and is therefore omitted here.

B.2 Detailed equivalence of the FE scheme with a neural network

Proof of Linear Homomorphism

For all $(\vec{x}, \vec{y}) \in \mathbb{Z}_p^n \times \mathbb{Z}_p^n$, and $(\vec{u}, \vec{v}) \in \mathbb{Z}_p^n \times \mathbb{Z}_p^n$, given an encryption of (\vec{x}, \vec{y}) under the public key $pk := (g_1^{\vec{s}}, g_2^{\vec{t}})$, one can efficiently compute an encryption of $(\vec{u}^\top \vec{x}, \vec{v}^\top \vec{y})$ under the public key $pk' := (g_1^{\vec{u}^\top \vec{s}}, g_2^{\vec{v}^\top \vec{t}})$. Indeed, given

$$\text{Enc}(pk, (\vec{x}, \vec{y})) := (g_1^\gamma, \{g_1^{\vec{a}_i}, g_2^{\vec{b}_i}\}_{i \in [n]}),$$

and $\vec{u}, \vec{v} \in \mathbb{Z}_p^n$, one can efficiently compute:

$$(g_1^\gamma, g_1^{\sum_{i \in [n]} u_i \cdot \vec{a}_i}, g_2^{\sum_{i \in [n]} v_i \cdot \vec{b}_i}),$$

which is $\text{Enc}(pk', (\vec{u}^\top \vec{x}, \vec{v}^\top \vec{y}))$, since:

$$\begin{aligned} \sum_{i \in [n]} u_i \cdot \vec{a}_i &= \sum_{i \in [n]} u_i \cdot (\mathbf{W}^{-1})^\top \begin{pmatrix} x_i \\ \gamma s_i \end{pmatrix} = (\mathbf{W}^{-1})^\top \begin{pmatrix} \sum_{i \in [n]} u_i \cdot x_i \\ \gamma \sum_{i \in [n]} u_i \cdot s_i \end{pmatrix} \\ &= (\mathbf{W}^{-1})^\top \begin{pmatrix} \vec{u}^\top \vec{x} \\ \gamma \vec{u}^\top \vec{s} \end{pmatrix}. \end{aligned}$$

Similarly, we have:

$$\sum_{i \in [n]} v_i \cdot \vec{b}_i = \sum_{i \in [n]} v_i \cdot \mathbf{W} \begin{pmatrix} y_i \\ -t_i \end{pmatrix} = \mathbf{W} \begin{pmatrix} \vec{v}^\top \vec{y} \\ -\vec{v}^\top \vec{t} \end{pmatrix}.$$

C Additional results

C.1 Influence of weight compression on the network performance

We show here that we can manage to compress significantly the network weights in order to have a very fast discrete logarithm without modifying the results and conclusions made throughout the article. The main and collateral model follow the same CNN structure as stated above, and the collateral accuracy is reported after 10 epochs of training.

Main accuracy with compression	$97.72 \pm 0.30 \%$
Collateral accuracy with compression	$55.27 \pm 0.41 \%$

Table 2: Impact of weight compression on the main and collateral accuracies

C.2 Influence of alpha during adversarial training

To choose the best value for α , we have chosen an output size of 4 which allows us to keep a very high main accuracy while reducing significantly the collateral one, as shown in Figure 5. We observe that the semi-adversarial training does not affect much the main accuracy for a large range of values for α , while its impact on the collateral accuracy is decisive. Figure 12 illustrates the role of α and justify our choice of $\alpha = 1.7$. For this experiment, we have chosen for both networks a simple feed forward with a hidden layer of 32 neurons.

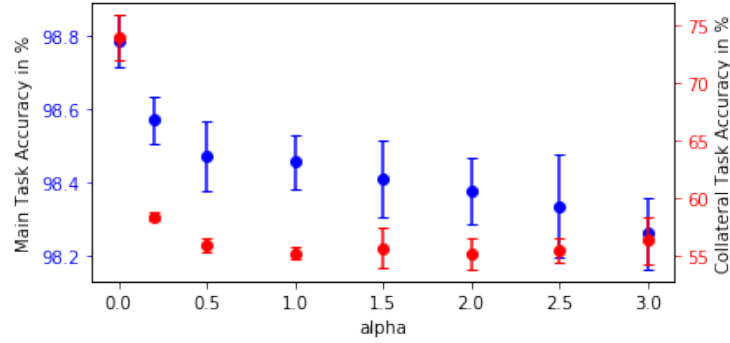


Figure 12: Trade-off between the main and collateral tasks accuracies as a function of α

D Security proof of our FE scheme

Proof. For any experiment Exp , adversary \mathcal{A} , and security parameter $\lambda \in \mathbb{N}$, we use the notation: $\text{Adv}_{\text{Exp}}(\mathcal{A}) := \Pr[1 \leftarrow \text{Exp}(1^\lambda, \mathcal{A})]$, where the probability is taken over the random coins of Exp and \mathcal{A} .

While we want to prove the security result in the real experiment Exp_0 , in which the adversary has to guess β , we slightly modify it into the hybrid experiment Exp_1 , described in 13: we write the matrix

$\text{Exp}_1(1^\lambda, \mathcal{A})$: $(\mathbb{G}_1, \mathbb{G}_2, p, g_1, g_2, e) \leftarrow \text{GGen}(1^\lambda), \vec{s}, \vec{t} \xleftarrow{\$} \mathbb{Z}_p^n$ $a, b, c, d \xleftarrow{\$} \mathbb{Z}_p$, set $\mathcal{PG} := (\mathbb{G}_1, \mathbb{G}_2, p, g_1^{ad-bc}, g_2, e)$ $\text{msk} := (\vec{s}, \vec{t}), \text{pk} := (\mathcal{PG}, g_1^{(ad-bc)\vec{s}}, g_2^{\vec{t}})$ $((\vec{x}^{(0)}, \vec{y}^{(0)}), (\vec{x}^{(1)}, \vec{y}^{(1)})) \leftarrow \mathcal{A}^{\text{KeyGen}(\text{msk}, \cdot)}(\text{pk})$ $\beta \xleftarrow{\$} \{0, 1\}, \gamma \xleftarrow{\$} \mathbb{Z}_p$ for all $i \in [n], \vec{a}_i := \begin{pmatrix} d & -c \\ -b & a \end{pmatrix} \begin{pmatrix} x_i^{(\beta)} \\ \gamma s_i \end{pmatrix}, \vec{b}_i := \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} y_i^{(\beta)} \\ -t_j \end{pmatrix}$ $ct := (g_1^{\gamma(ad-bc)}, \{g_1^{\vec{a}_i}, g_2^{\vec{b}_i}\}_{i \in [n]})$ $\beta' \leftarrow \mathcal{A}^{\text{KeyGen}(\text{msk}, \cdot)}(\text{pk}, ct)$ Return 1 if $\beta' = \beta$ and for all queried $f, f(\vec{x}^{(0)}, \vec{y}^{(0)}) = f(\vec{x}^{(1)}, \vec{y}^{(1)})$.	$\text{KeyGen}(\text{msk}, f)$: return $(g_2^{f(\vec{s}, \vec{t})}, f)$.
---	---

Figure 13: Experiment Exp_1 , for the proof of Theorem B.1.

$\mathbf{W} \xleftarrow{\$} \text{GL}_2$ used in the challenge ciphertext as $\mathbf{W} := \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, chosen from the beginning. Then

$$\mathbf{W}^{-1} = \frac{1}{ad-bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

The only difference with the IND-CPA security game as defined in Appendix A.2, is that we change the generator $g_1 \xleftarrow{\$} \mathbb{G}_1^*$ into g_1^{ad-bc} for $a, b, c, d \xleftarrow{\$} \mathbb{Z}_p$, which only changes the distribution of the game by a statistical distance of at most $\frac{3}{p}$ (this is obtained by computing the probability that $ad - bc = 0$ when $a, b, c, d \xleftarrow{\$} \mathbb{Z}_p$). Thus,

$$\text{Adv}_{\mathcal{A}}^{\text{FE}}(\lambda) = \text{Adv}_0(\mathcal{A}) \leq \text{Adv}_1(\mathcal{A}) + \frac{3}{p}.$$

Note that in Exp_1 , the public key, the challenge ciphertext and the functional decryption keys only contain group elements whose exponents are polynomials evaluated on random inputs (as opposed to $g_1^{\mathbf{W}^{-1}}$, for instance). This is going to be helpful for the next step of the proof, which uses the generic bilinear group model.

Next, we make the generic bilinear group model assumption, which intuitively says that no PPT adversary can exploit the structure of the bilinear group to perform better attacks than generic adversaries. That is, we have (with Exp_2 defined in 14):

$$\max_{\text{PPT } \mathcal{A}} (\text{Adv}_1(\mathcal{A})) = \max_{\text{PPT } \mathcal{A}} (\text{Adv}_2(\mathcal{A})).$$

In this experiment, we denote by \emptyset the empty list, by $\text{append}(L, x)$ the addition of an element x to the list L , and for any $i \in \mathbb{N}$, we denote by $L[i]$ the i 'th element of the list L if it exists (lists are indexed from index 1 on), or \perp otherwise.

Thus, it suffices to show that for any PPT adversary \mathcal{A} , $\text{Adv}_2(\mathcal{A})$ is negligible in λ . The experiment Exp_2 defined in Figure 14 falls into the general class of simple interactive decisional problems from [6, Definition 14]. Thus, we can use their master theorem [6, Theorem 7], which, for our particular case (setting the public key size $N := 2n + 2$, the key size $c = 1$, the ciphertext size $c^* := 4n + 1$, and degree $d = 6$ in [6, Theorem 7]) states that:

$$\text{Adv}_2(\mathcal{A}) \leq \frac{12 \cdot (6n + 3 + Q + Q')^2}{p},$$

where Q' is the number of queries to \mathcal{O}_{sk} , and Q is the number of group operations, that is, the number of calls to oracles \mathcal{O}_{add} and $\mathcal{O}_{\text{pair}}$, provided the following algebraic condition is satisfied:

$$\begin{aligned} & \{\mathbf{M} \in \mathbb{Z}_p^{(3n+2) \times (3n+Q'+1)} : \text{Eq}_0(\mathbf{M})\} \\ &= \{\mathbf{M} \in \mathbb{Z}_p^{(3n+2) \times (3n+Q'+1)} : \text{Eq}_1(\mathbf{M})\}, \end{aligned}$$

where for all \mathbf{M} , $b \in \{0, 1\}$,

$$\text{Eq}_b(\mathbf{M}) : \begin{pmatrix} 1 \\ (AD - BC)\vec{S} \\ (AD - BC)\Gamma \\ D\vec{x}^{(b)} - \Gamma C\vec{S} \\ -B\vec{x}^{(b)} + \Gamma A\vec{S} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 1 \\ \vec{T} \\ A\vec{y}^{(b)} - B\vec{T} \\ C\vec{y}^{(b)} - D\vec{T} \\ (f(\vec{S}, \vec{T}))_{f \in Q_{\text{sk}}} \end{pmatrix} = 0,$$

where the equality is taken in the ring $\mathbb{Z}_p[\vec{S}, \vec{T}, A, B, C, D, \Gamma]$, and 0 denotes the zero polynomial. Intuitively, this condition captures the security at a symbolic level: it holds for schemes that are not trivially broken. The latter means that computing a linear combination in the exponents of target group elements that can be obtained from pk, the challenge ciphertext, and functional decryption keys, does not break the security of the scheme. We prove this condition is satisfied in D.1 below. \square

Lemma D.1 (Symbolic Security) *For any $(\vec{x}^{(0)}, \vec{y}^{(0)}), (\vec{x}^{(1)}, \vec{y}^{(1)}) \in \mathbb{Z}_p^{2n}$, and any set $Q_{\text{sk}} \subseteq \mathcal{F}_{n, B_x, B_y, B_f}$ such that for all $f \in Q_{\text{sk}}$, $f(\vec{x}^{(0)}, \vec{y}^{(0)}) = f(\vec{x}^{(1)}, \vec{y}^{(1)})$, we have:*

$$\begin{aligned} & \{\mathbf{M} \in \mathbb{Z}_p^{(3n+2) \times (3n+Q'+1)} : \text{Eq}_0(\mathbf{M})\} \\ &= \{\mathbf{M} \in \mathbb{Z}_p^{(3n+2) \times (3n+Q'+1)} : \text{Eq}_1(\mathbf{M})\}, \end{aligned}$$

Exp₂(1^λ, A):
 $L_1 = L_2 = L_T := \emptyset, Q_{\text{sk}} := \emptyset, \vec{s}, \vec{t} \xleftarrow{\$} \mathbb{Z}_p^n, a, b, c, d \xleftarrow{\$} \mathbb{Z}_p, \text{append}(L_1, (ad - bc) \cdot \vec{s}),$
 $\text{append}(L_2, \vec{t}), \beta \xleftarrow{\$} \{0, 1\}$
 $((\vec{x}^{(0)}, \vec{y}^{(0)}), (\vec{x}^{(1)}, \vec{y}^{(1)})) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{add}}, \mathcal{O}_{\text{pair}}, \mathcal{O}_{\text{sk}}, \mathcal{O}_{\text{eq}}}(1^\lambda, p)$
 $\mathcal{O}_{\text{chal}}((\vec{x}^{(0)}, \vec{y}^{(0)}), (\vec{x}^{(1)}, \vec{y}^{(1)}))$
 $\beta' \leftarrow \mathcal{A}^{\mathcal{O}_{\text{add}}, \mathcal{O}_{\text{pair}}, \mathcal{O}_{\text{sk}}, \mathcal{O}_{\text{eq}}}(1^\lambda, p)$
 If $\beta = \beta'$, and for all $f \in Q_{\text{sk}}, f(\vec{x}^{(0)}, \vec{y}^{(0)}) = f(\vec{x}^{(1)}, \vec{y}^{(1)})$, output 1. Otherwise, output 0.

$\mathcal{O}_{\text{add}}(s \in \{1, 2, T\}, i, j \in \mathbb{N})$:
 $\text{append}(L_s, L_s[i] + L_s[j]).$

$\mathcal{O}_{\text{pair}}(i, j \in \mathbb{N})$:
 $\text{append}(L_T, L_1[i] \cdot L_2[j]).$

$\mathcal{O}_{\text{chal}}((\vec{x}^{(0)}, \vec{y}^{(0)}), (\vec{x}^{(1)}, \vec{y}^{(1)}))$:
 $\gamma \xleftarrow{\$} \mathbb{Z}_p, \text{append}(L_1, \gamma(ad - bc))$
 for all $i \in [n], \vec{a}_i := \begin{pmatrix} d & -c \\ -b & a \end{pmatrix} \begin{pmatrix} x_i^{(\beta)} \\ \gamma s_i \end{pmatrix}, \text{append}(L_1, \vec{a}_i), \vec{b}_i := \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} y_i^{(\beta)} \\ -t_i \end{pmatrix},$
 $\text{append}(L_2, \vec{b}_i).$

$\mathcal{O}_{\text{sk}}(f \in \mathcal{F}_{n, B_x, B_y, B_f})$:
 $\text{append}(L_2, f(\vec{s}, \vec{t})), Q_{\text{sk}} := Q_{\text{sk}} \cup \{f\}.$

$\mathcal{O}_{\text{eq}}(s \in \{1, 2, T\}, i, j \in \mathbb{N})$:
 Output 1 if $L_s[i] = L_s[j]$, 0 otherwise

Figure 14: Experiment Exp₂. Wlog. we assume no query contains indices $i, j \in \mathbb{N}$ that exceed the size of the involved lists.

where for all $\mathbf{M}, b \in \{0, 1\}$,

$$\text{Eq}_b(\mathbf{M}) : \begin{pmatrix} 1 \\ (AD - BC)\vec{S} \\ (AD - BC)\Gamma \\ D\vec{x}^{(b)} - \Gamma C\vec{S} \\ -B\vec{x}^{(b)} + \Gamma A\vec{S} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 1 \\ \vec{T} \\ A\vec{y}^{(b)} - B\vec{T} \\ C\vec{y}^{(b)} - D\vec{T} \\ (f(\vec{S}, \vec{T}))_{f \in Q_{\text{sk}}} \end{pmatrix} = 0,$$

where the equality is taken in the ring $\mathbb{Z}_p[\vec{S}, \vec{T}, A, B, C, D, \Gamma]$, and 0 denotes the zero polynomial.

Proof. Let $b \in \{0, 1\}$, and $\mathbf{M} \in \mathbb{Z}_p^{(3n+2) \times (3n+Q'+1)}$ that satisfies $\text{Eq}_b(\mathbf{M})$. We prove it also satisfies $\text{Eq}_{1-b}(\mathbf{M})$. To do so, we use the following rules:

Rule 1 : for all $P, Q, R \in \mathbb{Z}_p[\vec{S}, \vec{T}, A, B, C, D, \Gamma]$, with $\deg(P) \geq 1$, if $P \cdot Q + R = 0$ and R is not a multiple of P , then $Q = 0$ and $R = 0$.

Rule 2 : for all $P \in \mathbb{Z}_p[\vec{S}, \vec{T}, A, B, C, D, \Gamma]$, any variable X among the set $\{\vec{S}, \vec{T}, A, B, C, D, \Gamma\}$, and any $x \in \mathbb{Z}_p$, $P = 0$ implies $P(X := x) = 0$, where $P(X := x)$ denotes the polynomial P evaluated on $X = x$.

Evaluating $\text{Eq}_b(\mathbf{M})$ on $B = D = 0$ (using **Rule 2**), then using **Rule 1** on $P = \Gamma S_i T_j$ for all $i, j \in [n]$, we obtain that:

$$\mathbf{M}_{n+2+i} \begin{pmatrix} 0 \\ \vec{T} \\ \mathbf{0} \\ \mathbf{0} \\ (f(\vec{S}, \vec{T}))_{f \in Q_{\text{sk}}} \end{pmatrix} = 0,$$

where \mathbf{M}_{n+2+i} denotes the $n+2+i$ 'th row of \mathbf{M} .

Similarly, using **Rule 1** on $P = \Gamma A S_i T_j$ for all $i, j \in [n]$, we obtain that:

$$\mathbf{M}_{2n+2+i} \begin{pmatrix} 0 \\ \vec{T} \\ \mathbf{0} \\ \mathbf{0} \\ (f(\vec{S}, \vec{T}))_{f \in Q_{\text{sk}}} \end{pmatrix} = 0.$$

Thus, we have:

$$\forall \beta \in \{0, 1\} : \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\vec{x}^{(\beta)} - \Gamma C\vec{S} \\ -B\vec{x}^{(\beta)} + \Gamma A\vec{S} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 0 \\ \vec{T} \\ \mathbf{0} \\ \mathbf{0} \\ (f(\vec{S}, \vec{T}))_{f \in Q_{\text{sk}}} \end{pmatrix} = 0. \quad (1)$$

Using **Rule 1** on $P = (AD - BC)S_i B T_j$ for all $i, j \in [n]$ in the equation $\text{Eq}_b(\mathbf{M})$, we get that the coefficient $M_{i+1, n+1+j} = 0$ for all $i, j \in [n]$. Similarly, using **Rule 1** on $P = (AD - BC)S_i D T_j$ for all $i, j \in [n]$, we get $M_{i+1, 2n+1+j} = 0$ for all $i, j \in [n]$. Then, using **Rule 1** on $P = (AD - BC)\Gamma B T_j$ for all $j \in [n]$, we get $M_{n+2, n+1+j} = 0$ for all $j \in [n]$. Finally, using **Rule 1** on $P = (AD - BC)\Gamma D T_j$ for all $j \in [n]$, we get $M_{n+2, 2n+1+j} = 0$ for all $j \in [n]$. Overall, we obtain:

$$\forall \beta \in \{0, 1\} : \begin{pmatrix} 0 \\ (AD - BC)\vec{S} \\ (AD - BC)\Gamma \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\vec{y}^{(\beta)} - B\vec{T} \\ C\vec{y}^{(\beta)} - D\vec{T} \\ \mathbf{0} \end{pmatrix} = 0. \quad (2)$$

We write:

$$\begin{aligned} & \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\vec{x}^{(b)} - \Gamma C\vec{S} \\ -B\vec{x}^{(b)} + \Gamma A\vec{S} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\vec{y}^{(b)} - B\vec{T} \\ C\vec{y}^{(b)} - D\vec{T} \\ \mathbf{0} \end{pmatrix} \\ &= \sum_{i,j \in [n]} \begin{pmatrix} D x_i^{(b)} - \Gamma C S_i \\ -B x_i^{(b)} + \Gamma A S_i \end{pmatrix}^\top \\ & \times \left(m_{i,j}^{(1)} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + m_{i,j}^{(2)} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + m_{i,j}^{(3)} \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} + m_{i,j}^{(4)} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \right) \\ & \times \begin{pmatrix} A y_j^{(b)} - B T_j \\ C y_j^{(b)} - D T_j \end{pmatrix} \end{aligned}$$

Evaluating the equation $\text{Eq}_b(\mathbf{M})$ on $C = D = 0$ (by **Rule 2**), then using **Rule 1** on $P = \Gamma A B S_i T_j$ for all $i, j \in [n]$, we obtain $m_{i,j}^{(3)} = 0$ for all $i, j \in [n]$. Evaluating the equation $\text{Eq}_b(\mathbf{M})$ on $A = B = 0$ (by **Rule 2**), then using **Rule 1** on $P = \Gamma C D S_i T_j$ for all $i, j \in [n]$, we obtain $m_{i,j}^{(4)} = 0$

for all $i, j \in [n]$. Evaluating the equation $\text{Eq}_b(\mathbf{M})$ on $A = B = C = D = 1$ (using **Rule 2**), then using **Rule 1** on $P = \Gamma S_i T_j$ for all $i, j \in [n]$, using the fact that $m_{i,j}^{(3)} = m_{i,j}^{(4)} = 0$ and (1), we obtain $m_{i,j}^{(2)} = 0$ for all $i, j \in [n]$. Using **Rule 1** on $P = \Gamma(AD - BC)S_i T_j$ for all $i, j \in [n]$ in the equation $\text{Eq}_b(\mathbf{M})$, we obtain that for all $i, j \in [n]$,

$$m_{i,j}^{(1)} = \mathbf{M}_{n+2} \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ \mathbf{0} \\ (f_{i,j})_{f \in Q_{\text{sk}}} \end{pmatrix},$$

where \mathbf{M}_{n+2} is the $n + 2$ 'th row of \mathbf{M} .

Putting everything together, can write

$$\begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\vec{x}^{(b)} - \Gamma C\vec{S} \\ -B\vec{x}^{(b)} + \Gamma A\vec{S} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\vec{y}^{(b)} - B\vec{T} \\ C\vec{y}^{(b)} - D\vec{T} \\ \mathbf{0} \end{pmatrix}$$

as

$$\begin{aligned} & (AD - BC)\mathbf{M}_{n+2} \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ \mathbf{0} \\ (f(\vec{x}^{(b)}, \vec{y}^{(b)}) - \Gamma f(\vec{s}, \vec{t}))_{f \in Q_{\text{sk}}} \end{pmatrix} \\ &= (AD - BC)\mathbf{M}_{n+2} \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ \mathbf{0} \\ (f(\vec{x}^{(1-b)}, \vec{y}^{(1-b)}) - \Gamma f(\vec{s}, \vec{t}))_{f \in Q_{\text{sk}}} \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\vec{x}^{(1-b)} - \Gamma C\vec{S} \\ -B\vec{x}^{(1-b)} + \Gamma A\vec{S} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\vec{y}^{(b)} - B\vec{T} \\ C\vec{y}^{(b)} - D\vec{T} \\ \mathbf{0} \end{pmatrix} \end{aligned} \quad (3)$$

where we use the fact that for all $f \in Q_{\text{sk}}$, we have the equality $f(\vec{x}^{(b)}, \vec{y}^{(b)}) = f(\vec{x}^{(1-b)}, \vec{y}^{(1-b)})$.

Evaluating equation $\text{Eq}_b(\mathbf{M})$ on $A = B = D = 0$ (by **Rule 2**), then using **Rule 1** on $\Gamma S_i C$ for all $i \in [n]$, and using (1) and (3), we obtain that the coefficient $M_{n+2+i,1} = 0$ for all $i \in [n]$. Evaluating $\text{Eq}_b(\mathbf{M})$ on $B = C = D = 0$ (by **Rule 2**), then using **Rule 1** on $\Gamma S_i A$ for all $i \in [n]$, and using (1) and (3), we obtain that the coefficient $M_{2n+2+i,1} = 0$ for all $i \in [n]$. Thus, we have:

$$\forall \beta \in \{0, 1\} : \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\vec{x}^{(\beta)} - \Gamma C\vec{S} \\ -B\vec{x}^{(\beta)} + \Gamma A\vec{S} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 1 \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} = 0. \quad (4)$$

Evaluating equation $\text{Eq}_b(\mathbf{M})$ on $A = C = D = 0$ (by **Rule 2**), then using **Rule 1** on BT_j for all $i \in [n]$, and using (3), we obtain that the coefficient $M_{1,n+1+j} = 0$ for all $j \in [n]$. Evaluating $\text{Eq}_b(\mathbf{M})$ on $A = B = C = 0$ (by **Rule 2**), then using **Rule 1** on DT_j for all $j \in [n]$, and using (3), we obtain that the coefficient $M_{1,2n+1+j} = 0$ for all $j \in [n]$. Thus, we have:

$$\forall \beta \in \{0, 1\} : \begin{pmatrix} 1 \\ \mathbf{0} \\ 0 \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\vec{y}^{(\beta)} - B\vec{T} \\ C\vec{y}^{(\beta)} - D\vec{T} \\ \mathbf{0} \end{pmatrix} = 0. \quad (5)$$

Overall, we have:

$$\text{Eq}_b(\mathbf{M}) : \begin{pmatrix} 1 \\ (AD - BC)\vec{S} \\ (AD - BC)\Gamma \\ D\vec{x}^{(b)} - \Gamma C\vec{S} \\ -B\vec{x}^{(b)} + \Gamma A\vec{S} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 1 \\ \vec{T} \\ A\vec{y}^{(b)} - B\vec{T} \\ C\vec{y}^{(b)} - D\vec{T} \\ (f(\vec{S}, \vec{T}))_{f \in Q_{\text{sk}}} \end{pmatrix} = 0$$

which implies the following relation, under (1),(2),(4),(5)

$$\begin{aligned} & \begin{pmatrix} 1 \\ (AD - BC)\vec{S} \\ (AD - BC)\Gamma \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 1 \\ \vec{T} \\ \mathbf{0} \\ \mathbf{0} \\ (f(\vec{S}, \vec{T}))_{f \in Q_{\text{sk}}} \end{pmatrix} \\ & + \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\vec{x}^{(b)} - \Gamma C\vec{S} \\ -B\vec{x}^{(b)} + \Gamma A\vec{S} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\vec{y}^{(b)} - B\vec{T} \\ C\vec{y}^{(b)} - D\vec{T} \\ \mathbf{0} \end{pmatrix} = 0 \end{aligned}$$

and then, under (3)

$$\begin{aligned} & \begin{pmatrix} 1 \\ (AD - BC)\vec{S} \\ (AD - BC)\Gamma \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 1 \\ \vec{T} \\ \mathbf{0} \\ \mathbf{0} \\ (f(\vec{S}, \vec{T}))_{f \in Q_{\text{sk}}} \end{pmatrix} \\ & + \begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\vec{x}^{(1-b)} - \Gamma C\vec{S} \\ -B\vec{x}^{(1-b)} + \Gamma A\vec{S} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\vec{y}^{(1-b)} - B\vec{T} \\ C\vec{y}^{(1-b)} - D\vec{T} \\ \mathbf{0} \end{pmatrix} = 0. \end{aligned}$$

Under (1),(2),(4),(5), this implies

$$\text{Eq}_{1-b}(\mathbf{M}) : \begin{pmatrix} 1 \\ (AD - BC)\vec{S} \\ (AD - BC)\Gamma \\ D\vec{x}^{(1-b)} - \Gamma C\vec{S} \\ -B\vec{x}^{(1-b)} + \Gamma A\vec{S} \end{pmatrix}^\top \mathbf{M} \begin{pmatrix} 1 \\ \vec{T} \\ A\vec{y}^{(1-b)} - B\vec{T} \\ C\vec{y}^{(1-b)} - D\vec{T} \\ (f(\vec{S}, \vec{T}))_{f \in Q_{\text{sk}}} \end{pmatrix} = 0$$

□