# Reading in the Dark: Classifying Encrypted Digits with Functional Encryption

Edouard Dufour Sans[1,2], Romain Gay[1,2], and David Pointcheval[1,2]

[1] DIENS, École normale supérieure, CNRS,
PSL Research University, Paris, France
[2] INRIA
{edufoursans,rgay,david.pointcheval}@ens.fr

**Abstract.** As machine learning grows into a ubiquitous technology that finds many interesting applications, the privacy of data is becoming a major concern. This paper deals with machine learning and encrypted data. Namely, our contribution is twofold: we first build a new Functional Encryption scheme for quadratic multi-variate polynomials, which outperforms previous schemes. It enables the efficient computation of quadratic polynomials on encrypted vectors, so that only the result is in clear. We then turn to quadratic networks, a class of machine learning models, and show that their design makes them particularly suited to our encryption scheme. This synergy yields a technique for efficiently recovering a plaintext classification of encrypted data. Eventually, we prototype our construction and run it on the MNIST dataset to demonstrate practical relevance. We obtain 97.54% accuracy, with decryption and encryption taking few seconds.

**Keywords.** Machine Learning, Encrypted Data, Functional Encryption, Quadratic polynomials.

**Note.** An extended version has been published to NeuIPS 2019 [RDG+19], also available on arXiv.

## 1 Introduction

The growth of Machine Learning opens up many possibilities, but also raises challenges, particularly in the realm of privacy protection. Machine Learning finds applications in Finance, Medicine, and Cloud Computing, and in all cases, often seems to find itself in opposition to Cryptography. Recent works have proved this need not be the case: *Fully Homomorphic Encryption* (FHE) allows a server to perform computations on encrypted data, which means clients can delegate the evaluation of a Machine Learning model to a computationally more powerful server.

However, FHE only outputs an encrypted result (not the result in clear), which limits its application. The model's prediction may be of use to the server. When that is the case, FHE would require sending the encrypted result to the client for decryption (potentially with a Zero-Knowledge Proof that the computation was performed correctly when the client is concerned that the server might try to cheat), then having the client return the cleartext to the server (potentially with a Zero-Knowledge Proof certifying that decryption was performed correctly when the server is concerned that the client might try to cheat).

**For many practical applications, FHE is simply not the answer.** When the goal is to enable the party performing the computations to recover the result *in the clear*, and, necessarily, to provide some way for the owner of the data to limit which computations can be performed, one should look to *Functional Encryption* instead.

*Functional Encryption* (FE) [BSW11, O'N10] is a new paradigm for encryption which extends the "all-or-nothing" requirement of traditional Public Key Encryption in a much more flexible way. FE allows users to learn specific evaluations of a plaintext from the corresponding ciphertext: for any function $f$ from a class $\mathcal{F}$, a functional decryption key $\mathsf{dk}_f$ can be generated such that, given any ciphertext $c$ with underlying plaintext $x$, using $\mathsf{dk}_f$, a user can efficiently decrypt $c$ to obtain $f(x)$ without gaining any additional information about $x$.

Put simply, FE allows the user to control what is leaked about his data. Many applications such as spam filters, parental control, or targeted advertising, only require partial knowledge of the data. FE reconciles these useful applications with the need for privacy and confidentiality of the data, since only the relevant, processed information is revealed.

*Use cases for FE.* Consider the following scenario: Alice uses a secure email protocol *à la* PGP [Zim95], but which makes use of functional encryption. Bob uses Alice's public key to send her an email, which lands on Alice's email provider server. Alice gave the server keys that enable it to process the email and take appropriate actions without her being online. The server could learn how urgent the email is, and, accordingly decide to alert Alice. It could decide whether the message is spam and not even notify Alice in that case. It could even generate an automatic response encrypted under Bob's public key which it could send right away.

The aforementioned applications, while theoretically feasible, cannot realistically be implemented today: constructions of FE for all circuits are known but rely on non-standard assumptions and are extremely inefficient.

*Practical FE.* Abdalla *et al.* [ABDP15] suggested instead that we could construct FE schemes for limited classes of functions but that remained practical, and were based on standard assumptions. They introduced FE schemes for the Inner-Product functionality secure under the DDH assumption, making it the first FE functionality (besides different forms access control) for which practical schemes were proposed. More recently, at CRYPTO'2017, Baltico *et al.* [BCFG17] introduced schemes for *quadratic* functional encryption. One of those is only proven secure in the generic group model but is efficient enough that we can envision practical applications that make use of it.

### 1.1 Our Results

We demonstrate the practicality of quadratic FE in the context of image classification. After introducing and proving secure in the Generic Group Model our own Quadratic FE scheme which is slightly more efficient than that of [BCFG17], we train a classification model that tackles the standard MNIST dataset [LC10]. The model class —degree 2 polynomial networks with one hidden layer— is carefully chosen to enable optimizations on the functional decryption algorithm that bring the time required to recover a cleartext classification of an encrypted image down to a few seconds, in a prototype implementation developed using the Charm [AGM+13] framework with the PBC library [L+06] as back-end. We also tackle the problem of the need for a discrete logarithm computation, which arises in all but a few practical FE schemes: we both constrain the weights in our model to limit the size of the output domain and implement a Baby-Step Giant-Step Algorithm which involves more pre-computations than on-line computations. We store many discrete logarithms in a database which we fill once. This enables us to solve the (not too large) discrete logarithms efficiently.

### 1.2 Related Work

Our work is, to our knowledge, the first application of Functional Encryption to Machine Learning. We list and briefly summarize other works combining Cryptography and Machine Learning and an implementation of Functional Encryption.

*Classifying encrypted data via homomorphic encryption.* In [BLN14, DGBL+16, BMMP18] a user encrypts sensitive data using an homomorphic encryption scheme, sends it to the cloud, which can blindly classify it, using the homomorphic property of the encryption. But doing so, the cloud only obtains the encrypted result of the classification, and has to send it back to the user, who must decrypt it himself with his secret key. As in our work, the classifier used by the cloud is trained on plain data. The confidentiality of the data to be classified is guaranteed by the security of the homomorphic encryption scheme (in [BLN14, DGBL+16], they use the encryption scheme from [BLLN13], while [BMMP18] uses [CGGI16]). In fact, there is absolutely no leakage of information to the cloud (unlike our approach which leaks the result of the classification), since the cloud only sees the encrypted result. This, however, prevents different users from sharing sensitive data using the cloud, since only the user that encrypts has the key to decrypt. This is a pure outsourcing of computation: the user has to trust the server on the correctness of the computation (unless costly proofs are added) and the server cannot immediately base any further computations on the classification result, since it does not learn it.

An other work [BPTG15] considers the setting where the model itself, while trained on plain data, has to remain private from the persons classifying the encrypted data. They build efficient, specialized 2-party protocols for the core functions used in most common classifiers (such as linear, naive Bayes, or decision tree classifiers), using homomorphic encryption and garbled circuits.

*Learning on encrypted data.* While our work considers classifying encrypted data, using a classifier trained on plain data, [NWI+13] considers learning a linear curve that best fits the encrypted training data. The construction reveals the linear curve in the clear, and it can then be used for prediction on new data, but does not reveal any further information on the training data. This uses homomorphic encryption [Pai99], and garbled circuits [Yao86]. In [LP00], the authors build an optimized 2-party protocol for learning a decision tree from private databases.

*Implementations of FE schemes.* [KLM+18] implements a function-hiding FE for inner product. This is a private-key scheme where functional decryption keys decrypt an inner product of an encrypted vector, without revealing their underlying functions. Their source code, which also uses the Charm framework, is available on GitHub at `https://github.com/kevinlewi/fhipe`.

## 2 Preliminaries

### 2.1 Bilinear Groups

Our FE scheme uses bilinear groups (also known as pairing groups), whose use in cryptography has been introduced by [BF03, Jou04]. More precisely, we denote by $\mathsf{GGen}$ a PPT algorithm that on input $1^\lambda$ returns a description $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, p, g_1, g_2, e)$ of an asymmetric bilinear group, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are cyclic groups of prime order $p$ (for a $2\lambda$-bit prime $p$) and $g_1$ and $g_2$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. The application $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an admissible pairing, which means that it is efficiently computable, non-degenerated, and bilinear: $e(g_1^\alpha, g_2^\beta) = e(g_1, g_2)^{\alpha\beta}$ for any scalars $\alpha, \beta \in \mathbb{Z}_p$. We thus define $g_T := e(g_1, g_2)$ which spans the group $\mathbb{G}_T$ of prime order $p$.

For the sake of clarity, for any $s \in \{1, 2, T\}$, $n \in \mathbb{N}$, and vector $\boldsymbol{u} := \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} \in \mathbb{Z}_p^n$, we denote by $g_s^{\boldsymbol{u}} := \begin{pmatrix} g_s^{u_1} \\ \vdots \\ g_s^{u_n} \end{pmatrix} \in \mathbb{G}_s^n$. In the same vein, for any vectors $\boldsymbol{u} \in \mathbb{Z}_p^n, \boldsymbol{v} \in \mathbb{Z}_p^n$, we denote by $e(g_1^{\boldsymbol{u}}, g_2^{\boldsymbol{v}}) = \prod_{i=1} e(g_1, g_2)^{u_i \cdot v_i} = e(g_1, g_2)^{\boldsymbol{u} \cdot \boldsymbol{v}} \in \mathbb{G}_T$, since $\boldsymbol{u} \cdot \boldsymbol{v}$ denotes the inner product between the vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, that is: $\boldsymbol{u} \cdot \boldsymbol{v} := \sum_{i=1}^n u_i v_i$.

### 2.2 Functional Encryption

We recall the definition of Functional Encryption, originally defined in [BSW11, O'N10].

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

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

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

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

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

*Perfect correctness.* For all $x \in \mathcal{X}$, $f \in \mathcal{F}$,

$$\Pr[\mathsf{Dec}(\mathsf{dk}_f, \mathsf{ct}) = f(x)] = 1,$$

where the probability is taken over $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{SetUp}(1^\lambda \mathcal{F})$, $\mathsf{dk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$.

*IND-CPA security.* For any stateful adversary $\mathcal{A}$, and any functional encryption scheme $\mathsf{FE}$, we define the following advantage.

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

with the restriction that all queries $f$ that $\mathcal{A}$ makes to key generation algorithm $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ must satisfy $f(x_0) = f(x_1)$. We say $\mathsf{FE}$ is IND-CPA secure if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FE}}(\lambda) = \mathrm{negl}(\lambda)$.

## 3 Functional Encryption for Quadratic Polynomials

Here we build an efficient FE scheme (described in Fig. 2) for the set of functions defined, for all $n, B_x, B_y, B_f \in \mathbb{N}^*$, as $\mathcal{F}_{n,B_x,B_y,B_f} = \{f : [-B_x, B_x]^n \times [-B_y, B_y]^n \to \mathbb{Z}\}$ where the functions $f \in \mathcal{F}_{n,B_x,B_y,B_f}$ are described as a set of bounded coefficients $\{f_{i,j} \in [-B_f, B_f]\}_{i,j\in[n]}$, and for all vectors $\boldsymbol{x} \in [-B_x, B_x]^n$, $\boldsymbol{y} \in [-B_y, B_y]$, we have:

$$f(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i,j\in[n]} f_{i,j} x_i y_j.$$

It relies on prime-order, asymmetric, bilinear groups (see Section 2.1), and is proven secure in the generic group model. We compare its efficiency with previous quadratic FE schemes in Fig. 1. Note that the efficiency of the decryption can be further optimized for those quadratic polynomials used that are relevant to our application (see Section 4).

For the encryption phase, $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ are not explicitly computed, but $g_1^{\boldsymbol{a}_i}$ and $g_2^{\boldsymbol{b}_i}$ can be computed from the group elements $g_1^{\boldsymbol{s}}$, $g_2^{\boldsymbol{t}}$, and the scalars in $\mathbf{W}$, $\gamma$, $x_i$, and $y_i$.

| FE scheme | $ct$ | $\mathsf{dk}_f$ | Dec | Assumption |
|---|---|---|---|---|
| BCFG 17 [BCFG17, 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 |
| BCFG 17 [BCFG17, 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 |

**Fig. 1.** Performance comparison of public-key FE for quadratic polynomials. $E_1$ and $P$ denote exponentiation in $\mathbb{G}_1$ and pairing evaluation, respectively. We ignore the description of the function $f$ in $\mathsf{dk}_f$. Decryption additionally requires solving a discrete logarithm. Since this computational overhead is the same for all schemes, we omit it here.

$\underline{\mathsf{SetUp}(1^\lambda, \mathcal{F}_{n,B_x,B_y,B_f})}$:
$\mathcal{PG} := (\mathbb{G}_1, \mathbb{G}_2, p, g_1, g_2, e) \leftarrow \mathsf{GGen}(1^\lambda)$, $\boldsymbol{s}, \boldsymbol{t} \xleftarrow{\$} \mathbb{Z}_p^n$, $\mathsf{msk} := (\boldsymbol{s}, \boldsymbol{t})$, $\mathsf{pk} := \left(\mathcal{PG}, g_1^{\boldsymbol{s}}, g_2^{\boldsymbol{t}}\right)$
Return $(\mathsf{pk}, \mathsf{msk})$.

$\underline{\mathsf{Enc}\left(\mathsf{pk}, (\boldsymbol{x}, \boldsymbol{y})\right)}$:
$\gamma \xleftarrow{\$} \mathbb{Z}_p$, $\mathbf{W} \xleftarrow{\$} \mathsf{GL}_2$, for all $i \in [n]$, $\boldsymbol{a}_i := (\mathbf{W}^{-1})^\top \begin{pmatrix} x_i \\ \gamma s_i \end{pmatrix}$, $\boldsymbol{b}_i := \mathbf{W} \begin{pmatrix} y_i \\ -t_i \end{pmatrix}$
Return $\mathsf{ct} := \left(g_1^\gamma, \{g_1^{\boldsymbol{a}_i}, g_2^{\boldsymbol{b}_i}\}_{i\in[n]}\right) \in \mathbb{G}_1 \times (\mathbb{G}_1^2 \times \mathbb{G}_2^2)^n$

$\underline{\mathsf{KeyGen}(\mathsf{msk}, f)}$:
Return $\mathsf{dk}_f := \left(g_2^{f(\boldsymbol{s},\boldsymbol{t})}, f\right) \in \mathbb{G}_2 \times \mathcal{F}_{n,B_x,B_y,B_f}$.

$\underline{\mathsf{Dec}\left(\mathsf{pk}, \mathsf{ct} := \left(g_1^\gamma, \{g_1^{\boldsymbol{a}_i}, g_2^{\boldsymbol{b}_i}\}_{i\in[n]}\right), \mathsf{dk}_f := \left(g_2^{f(\boldsymbol{s},\boldsymbol{t})}, f\right)\right)}$:
$out := e(g_1^\gamma, g_2^{f(\boldsymbol{s},\boldsymbol{t})}) \cdot \prod_{i,j\in[n]} e\left(g_1^{\boldsymbol{a}_i}, g_2^{\boldsymbol{b}_i}\right)^{f_{i,j}}$
Return $\log(out) \in \mathbb{Z}$.

**Fig. 2.** Our functional encryption scheme for quadratic polynomials

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

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

since

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

Thus, we have:

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

**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 [Mau05], 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 [Nec94, Sho97].

We prove security in two steps: first, we use a master theorem from [BCFG17] 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 [ABGW17]).

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 [BCFG17, Definition 10], and previously in [BFF+14]). 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 [BCFG17], we prove the security in the symbolic model (cf. Lemma 3), which simply consists of checking that an algebraic condition on the scheme in satisfied.

**Theorem 2 (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 Fig. 2, we have, in the generic bilinear group model:*

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

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

Since this proof is not the main result of this paper, it is postponed to the appendix.

*Linear homomorphism.* Our FE scheme (Fig. 2) enjoys the property that the encryption algorithm is linearly homomorphic with respect to both the plaintext and the public key. Namely, for all $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{Z}_p^n \times \mathbb{Z}_p^n$, and $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{Z}_p^n \times \mathbb{Z}_p^n$, given an encryption of $(\boldsymbol{x}, \boldsymbol{y})$ under the public key $\mathsf{pk} := (g_1^{\boldsymbol{s}}, g_2^{\boldsymbol{t}})$, one can efficiently compute an encryption of $(\boldsymbol{u}^\top \boldsymbol{x}, \boldsymbol{v}^\top \boldsymbol{y})$ under the public key $\mathsf{pk}' := (g_1^{\boldsymbol{u}^\top \boldsymbol{s}}, g_2^{\boldsymbol{v}^\top \boldsymbol{t}})$. Indeed, given

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

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

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

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

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

Similarly, we have:

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

This is of particular interest for functions $f \in \mathcal{F}_{n,B_x,B_y,B_f}$ such that for all $\boldsymbol{x} \in [-B_x, B_x]$, $\boldsymbol{y} \in [-B_y, B_y]$,

$$f(\boldsymbol{x}, \boldsymbol{y}) = (\mathbf{U}\boldsymbol{x})^\top \mathbf{M} (\mathbf{V}\boldsymbol{y}),$$

where $\mathbf{U} \in \mathbb{Z}_p^{d\times n}$ and $\mathbf{V} \in \mathbb{Z}_p^{d\times n}$ are projection matrices, for $d < n$, and $\mathbf{M} \in \mathbb{Z}_p^{d\times d}$, since decryption first computes the encryption of $(\mathbf{U}\boldsymbol{x}, \mathbf{V}\boldsymbol{y})$ by linear homomorphism, then applies the decryption algorithm on ciphertexts whose underlying plaintexts are vectors of dimensions $d$. This requires $2dn$ exponentiations in $\mathbb{G}_1$, $2dn$ exponentiations in $\mathbb{G}_2$, and $2d^2$ pairing computations, as opposed to $2n^2$ pairing evaluations for the naive decryption: this is a major efficiency improvement for small $d$, which is the case for the functions we are using to classify encrypted data, as explained in Section 4.

*Computing the discrete logarithm for decryption.* Our decryption requires computing discrete logarithms of elements of $\mathbb{G}_T$, in base $e(g_1, g_2)$, which is independent of the ciphertext and the functional decryption key used to decrypt. Thus, to speed decryption up, we can pre-compute a table of discrete logarithm values in $\mathbb{G}_T$ which is accessed during every decryption. See Section 5 for more details on the discrete logarithm computations. Note that previously implemented schemes, such as [KLM+18], do not satisfy this property, and thus need to compute the discrete logarithm from scratch with every new decryption.

## 4 Choosing a Model

We solve the challenging task of finding a model that is both accurate for classifying data and efficiently implementable by state of the art FE schemes.

A natural choice for its simplicity is to use a linear classifier, since efficient FE for linear functions exists [ABDP15, ALS16]. However, linear classifiers achieve limited accuracy when attempting to classify data (TensorFlow's tutorial [ten] claims 92% accuracy on MNIST dataset).

This unsatisfactory performance justifies the use of richer models, as permitted by our FE scheme for quadratic polynomials (introduced in Section 3). We want to classify data that can be represented as a vector $\boldsymbol{x} \in [0, B]^n$ for some $B, n \in \mathbb{N}$ (in the case of the MNIST dataset, the size $B = 255$, and the dimension $n = 784$). In the following, we build models $(f_i)_{i\in[\ell]}$ for each label $i \in [\ell]$, such that our prediction for the class of $\boldsymbol{x} \in [0, B]^n$ is $\underset{i\in[\ell]}{\operatorname{argmax}} f_i(\boldsymbol{x})$.

*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 the integers (see paragraph below), such that $f_i(\boldsymbol{x}) = \boldsymbol{x}^\top \mathbf{Q}_i \boldsymbol{x}$, $\forall i \in [\ell]$. This is a very powerful model with a lot of parameters: $\ell n^2$! In the case of MNIST ($n = 784$), the training set is arguably too small to make use of such a large number of parameters, and the resulting number of pairings to compute ($2 \times 784^2$ by reusing the pairings as explained below) would be unreasonably large.

*Projection and quadratic polynomial on $\mathbb{R}^d$.* To reduce the number of pairing computations, we first project the input vector from $\mathbb{R}^n$ onto $\mathbb{R}^d$ for a well chosen $d < n$, and we apply the quadratic polynomials on the projected vectors. We can do this thanks to our scheme's linear homomorphism (see Section 3). This means that we learn $\mathbf{P} \in \mathbb{R}^{n\times d}$ and $(\mathbf{Q}_i)_{i\in[\ell]} \in \left(\mathbb{R}^{d\times d}\right)^\ell$, and our model is $f_i(\boldsymbol{x}) = (\mathbf{P}\boldsymbol{x})^\top \mathbf{Q}_i (\mathbf{P}\boldsymbol{x})$, $\forall i \in [\ell]$. Notice that the same $\mathbf{P}$ is used in all of the $f_i$, so we need only compute $\mathbf{P}\boldsymbol{x}$ once, and the number of pairings is reduced to $2\ell d^2$. 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.

*Adding a bias term.* Our model would be more general if it were expanded from $f_i(\boldsymbol{x}) = (\mathbf{P}\boldsymbol{x})^\top \mathbf{Q}_i(\mathbf{P}\boldsymbol{x})$ to $f_i(\boldsymbol{x}) = (\mathbf{P}\boldsymbol{x}+\boldsymbol{b})^\top \mathbf{Q}_i(\mathbf{P}\boldsymbol{x}+\boldsymbol{b})$ for $\boldsymbol{b} \in \mathbb{Z}_p^d$. We achieve something equivalent by systematically adding a 1 at the beginning of $\boldsymbol{x}$ when encrypting it, effectively operating on $\begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{pmatrix}$.

*Degree 2 polynomial network, with one hidden layer.* To further reduce the number of pairings, we actually limit ourselves to diagonal matrices, we thus rename $\mathbf{Q}_i$ to $\mathbf{D}_i$. We find that the gain in efficiency associated with only computing $2d$ pairings (since the $\mathbf{D}_i$ are diagonal, they contain at most $d$ non-zero entries) is worth the small cost in accuracy. The resulting model is a polynomial network of degree 2 with one hidden layer of $d$ neurons: the activation function is the square.

*Rounding onto $\mathbb{Z}_p$.* Our encryption scheme operates on elements of $\mathbb{Z}_p$, so we need to round our model before we can use it on encrypted data. This does not significantly affect our accuracy.

Our final model can thus be written as

$$f_i(\boldsymbol{x}) = (\mathbf{P}\boldsymbol{x}')^\top \mathbf{D}_i(\mathbf{P}\boldsymbol{x}'), \forall i \in [\ell],$$

where $\boldsymbol{x}' = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{pmatrix}$. In the remainder of this paper we simply use $\boldsymbol{x}$ to denote $\boldsymbol{x}'$ when evaluating our model on an input $\boldsymbol{x}$.

We present the decryption algorithm that is optimized for our particular choice of model in Fig. 3.

$\underline{\mathsf{OptDec}\,(\mathsf{pk}, \mathsf{ct}, \mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_\ell})}$:

- Parse $\mathsf{ct} := \left(g_1^\gamma, \{g_1^{\boldsymbol{a}_i}, g_2^{\boldsymbol{b}_i}\}_{i\in[n]}\right)$,
where for all $i \in [n]$, $\boldsymbol{a}_i := (\mathbf{W}^{-1})^\top \begin{pmatrix} x_i \\ \gamma s_i \end{pmatrix}$, $\boldsymbol{b}_i := \mathbf{W} \begin{pmatrix} x_i \\ -t_i \end{pmatrix}$.
- For all $i \in [\ell]$, parse $\mathsf{sk}_{f_i} := \left(g_2^{f_i(\boldsymbol{s},\boldsymbol{t})}, f_i\right)$,
where $f_i(\boldsymbol{s},\boldsymbol{t}) := (\mathbf{P}\boldsymbol{s})^\top \mathbf{D}_i(\mathbf{P}\boldsymbol{t})$
for some fixed matrix $\mathbf{P} \in \mathbb{Z}_p^{n\times d}$,
and $\mathbf{D}_i := \begin{pmatrix} \delta_{i,1} & & \\ & \ddots & \\ & & \delta_{i,d} \end{pmatrix} \in \mathbb{Z}_p^{d\times d}$ for each label $i \in [\ell]$.
- We write $\mathbf{P} := \begin{pmatrix} \boldsymbol{p}_1^\top \\ \vdots \\ \boldsymbol{p}_d^\top \end{pmatrix}$, where for all $i \in [d]$, $\boldsymbol{p}_i^\top \in \mathbb{Z}_p^{1\times n}$ is the $i$'th row of $\mathbf{P}$.
For all $i \in [d]$, compute $e_i := e(g_1^{\boldsymbol{\alpha}_i}, g_2^{\boldsymbol{\beta}_i}) \in \mathbb{G}_T$
where $\boldsymbol{\alpha}_i := (\mathbf{W}^{-1})^\top \begin{pmatrix} \boldsymbol{p}_i^\top \boldsymbol{x} \\ \gamma \cdot \boldsymbol{p}_i^\top \boldsymbol{s} \end{pmatrix}$ and $\boldsymbol{\beta}_i := \mathbf{W} \begin{pmatrix} \boldsymbol{p}_i^\top \boldsymbol{x} \\ -\boldsymbol{p}_i^\top \boldsymbol{t} \end{pmatrix}$
- For all labels $i \in [\ell]$, compute $out_i := e(g_1^\gamma, g_2^{f_i(\boldsymbol{s},\boldsymbol{t})}) \cdot \prod_{j\in[d]} e_j^{\delta_{i,j}}$.
Return $\{\log(out_i) \in \mathbb{Z}\}_{i\in[\ell]}$.

**Fig. 3.** Optimized decryption algorithm, for quadratic polynomials of the form $f_i(\boldsymbol{x},\boldsymbol{x}) := (\mathbf{P}\boldsymbol{x})^\top \mathbf{D}_i(\mathbf{P}\boldsymbol{x})$ with $\mathbf{P} \in \mathbb{Z}_p^{n\times d}$, and $\mathbf{D}_i \in \mathbb{Z}_p^{d\times d}$ is a diagonal matrix for all labels $i \in [\ell]$. Its performance is: $2nd(E_1+E_2)+(\ell+2d)P+\ell d E_T + \ell \cdot \mathsf{dlog}$, where $E_1$, $E_T$ denote exponentiation in $\mathbb{G}_1$, $\mathbb{G}_T$ respectively, $P$ denotes a pairing evaluation, and $\mathsf{dlog}$ denotes the time needed to solve the discrete logarithm.

## 5 Implementation and Results

We train a polynomial network classifier in TensorFlow [AAB+15], and use it in conjunction with our FE scheme from Section 3, which we implement in Python, using the Charm framework [AGM+13] with the Pairing-Based Cryptography (PBC) library [L+06] as back-end. Our code will be uploaded to a public GitHub repository to which we will provide a link in the final version of this paper.

*Training a classifier in TensorFlow.* We follow a rather standard procedure on the Machine Learning side of the implementation. We describe our model in TensorFlow and train a classifier using the Adam optimization algorithm [KB14]. We use $\ell 2$-regularization to limit overfitting and adjust the hyperparameters using a validation set of 5000 labeled images. We train the final model on the full training set (60000 labeled images), and scale it so the largest scalars in absolute value of $\mathbf{P}$ and $(\mathbf{D}_i)_i$ are 15 and 30, respectively, before rounding it. Rounding it down to a small value is crucial to the efficiency of the final scheme. The resulting integer model achieves 97.54% accuracy on the test set of 10000 labeled images. We give a graphical representation of the confusion matrix in Fig. 4: each row represents a manuscript digit to be classified, and each column represents a classification result.

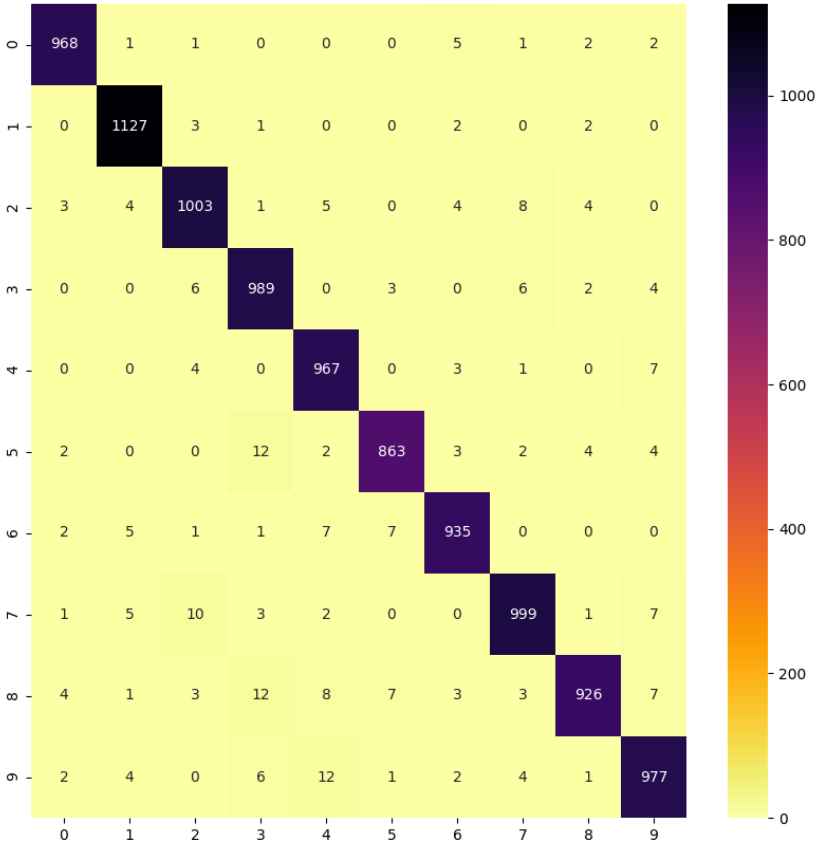


**Fig. 4.** Confusion matrix describing our performance on the MNIST dataset. Each row represents a manuscript digit to be classified, and each column represents a classification result.

*Implementing the scheme in Charm.* We use an asymmetric, prime-order, bilinear group: curve MNT159 [MNT01], which provides 80 bits of security. We essentially follow the description of the scheme given in Section 3, except for the decryption algorithm, which we optimized using our insights from Section 4, as described in Fig. 3. To gain more efficiency, we batch the computation of exponentiations. Namely, the decryption algorithm requires exponentiating the same group element by many scalars. Thus, we can re-use the exponentiations of the group element by the powers of two, used in the first step of the square and multiply algorithm, for all exponentiations[3].

[3] see https://en.wikipedia.org/wiki/Exponentiation_by_squaring for a description of the square and multiply algorithm

For encryption, we also need to compute exponentiations of a group element by many *small* scalars (in our case, the scalars are comprised between 0 and 255). We compute all the exponentiations for exponents between 0 and 255 once, and then, access them directly instead of actually computing new exponentiations. Doing so improves efficiency, and has the advantage of not introducing obvious timing attacks on the encryption procedure.

*Solving the discrete logarithms.* The computation of the 10 discrete logarithms (one for each label) can be prohibitive in terms of computation time, as the scores computed by our classifier can be quite large. We confront this issue by precomputing the giant steps of the Baby Step Giant Step algorithm during the setup. As mentioned in Section 3, the discrete logarithms we have to compute are always relative to the same base: $e(g_1, g_2)$. We can thus store a large amount of exponents of $e(g_1, g_2)$, which significantly speeds decryption up, at the cost of a larger usage of memory. We store pairs of integers matching the hash of a group element to its discrete logarithm in a PostgreSQL database. In our tests, we chose bounds based on the maximum scores our classifier gives on the MNIST dataset [LC10], by evaluating it on the plaintext. We allow at most $2^{13}$ baby steps for each discrete logarithm in the online phase, which requires 2.81 seconds on average, for 1.8 GB of storage. We provide a method giving loose bounds on the scores output by a given model, and from it, we estimate that computing the full database while requiring the same number of baby steps would require storing 26.3 GB. This can be prohibitive for an individual user, but should not be a problem for production software companies, as we envisioned when listing potential applications.

We give the average runtime for encryption, functional key generation, and decryption below. They were obtained using a 2.60GHz Intel Core i5-6440HQ CPU and 8GB of RAM. We break down the decryption phase into an evaluation phase (which covers exponentiations and pairings) and a discrete logarithm phase (whose runtime is independent of that of the latter, and can be reduced to arbitrarily short durations at the cost of storing an ever larger database of precomputations). We stress that the latter accounts for all 10 discrete logarithms. While we chose to demonstrate our techniques on a classification problem with 10 classes, they could also be applied to a regression problem, and in that case only one discrete logarithm would need to be retrieved.

| | |
|---|---|
| Average encryption time | 8.1s |
| Average evaluation time | 1.5s |
| Average discrete logarithms time | 3.3s |
| Average functional key generation time | 8ms |

## 6 Conclusion

In this work, we have proposed an efficient Functional Encryption scheme for the evaluation of multivariate quadratic polynomials. It outperforms all previous schemes. This opens up a path to richer classification models from Machine Learning. Thanks to our new FE scheme, one can indeed publicly classify encrypted data: given the functional decryption key, anyone can accurately predict the label that describes an encrypted digit, within just a few seconds, without being able to fully decrypt the ciphertext.

However, one can think to many improvements for better efficiency or more functionalities. Our implementation could indeed be improved in the following ways:

- Using faster languages and frameworks, such as C with direct calls to the PBC library [L+06].
- Training Machine Learning models that can be turned into integers with even smaller bounds, thereby greatly decreasing the required number of discrete logarithms.
- Using finer algorithms to estimate the bounds between which we must precompute discrete logarithms, perhaps by borrowing techniques from the field of Mathematical Optimization.
- Tackling different datasets, that might not be used as benchmarks like MNIST but that might have practical relevance when it comes to protecting privacy. Note that while this work focused entirely on classification problems, our scheme can readily be used for regression tasks, and those only require solving one discrete logarithm.

We also list several open problems:

- Combining the previous FHE-based approach, and our FE-based approach, to perform both the learning and classification on encrypted data, for better privacy.

- Designing and implementing efficient FE schemes for richer classes of functions, that capture more powerful machine learning algorithms. From an efficiency viewpoint, it would be interesting to have an efficient FE for large inputs, and, in particular, without the need to solve a discrete logarithm. This only exists for a restricted class of functions, namely, inner products [ABDP15, ALS16].
- Building efficient FE for unbounded size inputs, which would provide a solution to the problem of email filtering we mentioned in Section 1.
- Designing efficient function hiding FE schemes, that is, where functional decryption keys hide their underlying function. This is useful for many scenarios where the person doing the classification of encrypted data should not learn the classifier itself. Such schemes only exist for inner products [BJK15, DDM16, TAO16, KLM+18].

FE is a recent cryptographic primitive, yet this paper shows that it could already be used in practice. This is encouraging, because FE allows just the type of controlled access to data that is useful in many practical scenarios, and that current tools fail to provide. We hope this work inspires further contributions applying FE to real-world problems.

## Acknowledgments.

The authors thank Francis Bach for helpful discussions and comments.
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

AAB+15. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

ABDP15. 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.

ABGW17. 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 17*, pages 647–664. ACM Press, October / November 2017.

AGM+13. 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.

ALS16. 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.

BCFG17. 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.

BF03. Dan Boneh and Matthew K. Franklin. Identity based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

BFF+14. 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.

BJK15. Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. Function-hiding inner product encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 470–491. Springer, Heidelberg, November / December 2015.

BLLN13. Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In Martijn Stam, editor, *Cryptography and Coding*, pages 45–64, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

BLN14. Joppe W. Bos, Kristin E. Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data. *Journal of Biomedical Informatics*, 50:234–243, 2014.

BMMP18. Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 483–512. Springer, Heidelberg, August 2018.

BPTG15. Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, 2015.

BSW11. 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.

CGGI16. 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, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2016.

DDM16. Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Functional encryption for inner product with full function privacy. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 164–195. Springer, Heidelberg, March 2016.

DGBL+16. 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.

Jou04. Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, September 2004.

KB14. Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

KLM+18. 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.

L+06. Ben Lynn et al. The pairing-based cryptography library. *Internet: crypto.stanford.edu/pbc/ [Mar. 27, 2013]*, 2006.

LC10. Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

LP00. Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 36–54. Springer, Heidelberg, August 2000.

Mau05. 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.

MNT01. Atsuko Miyaji, Masaki Nakabayashi, and Shunzou TAKANO. New explicit conditions of elliptic curve traces for fr-reduction, 2001.

Nec94. V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.

NWI+13. Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE Symposium on Security and Privacy*, pages 334–348, 2013.

O'N10. Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. https://eprint.iacr.org/2010/556.

Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

RDG+19. Théo Ryffel, Edouard Dufour-Sans, Romain Gay, Francis Bach, and David Pointcheval. Partially Encrypted Machine Learning using Functional Encryption. In *Advances in Neural Information Processing Systems (NeurIPS 2019)*, Vancouver, Canada, 2019.

Sho97. 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.

TAO16. Junichi Tomida, Masayuki Abe, and Tatsuaki Okamoto. Efficient functional encryption for inner-product values with full-hiding security. In *International Conference on Information Security*, pages 408–425. Springer, 2016.

ten. Mnist for ml beginners | tensorflow.

Yao86. A. C. C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, Oct 1986.

Zim95. Philip R Zimmermann. *The official PGP user's guide*. MIT press, 1995.

## A Security proof of our FE scheme

**Theorem 2 (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 Fig. 2, we have, in the generic bilinear group model:*

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

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

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

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

**Fig. 5.** Experiment $\mathsf{Exp}_1$, for the proof of Theorem 2.

While we want to prove the security result in the real experiment $\mathsf{Exp}_0$, in which the adversary has to guess $\beta$, we slightly modify it into the hybrid experiment $\mathsf{Exp}_1$, described in Fig. 5: we write the matrix $\mathbf{W} \xleftarrow{\$} \mathsf{GL}_2$ 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 Section 2.2, 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,

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

Note that in $\mathsf{Exp}_1$, 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, weh have (wih $\mathsf{Exp}_2$ defined in Fig. 6):

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

In this experiment, we denote by $\emptyset$ the empty list, by $\mathsf{append}(L, x)$ 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 $\mathcal{A}$, $\mathsf{Adv}_2(\mathcal{A})$ is negligible in $\lambda$. The experiment $\mathsf{Exp}_2$ defined in Fig. 6 falls into the general class of simple interactive decisional problems from [BCFG17,

$\underline{\mathsf{Exp}_2(1^\lambda, \mathcal{A})}$:
$L_1 = L_2 = L_T := \emptyset$, $Q_{\mathsf{sk}} := \emptyset$, $\boldsymbol{s}, \boldsymbol{t} \xleftarrow{\$} \mathbb{Z}_p^n$, $a, b, c, d \xleftarrow{\$} \mathbb{Z}_p$, $\mathsf{append}(L_1, (ad-bc)\cdot\boldsymbol{s})$, $\mathsf{append}(L_2, \boldsymbol{t})$, $\beta \xleftarrow{\$} \{0,1\}$
$\left((\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}), (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)})\right) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{add}}, \mathcal{O}_{\mathsf{pair}}, \mathcal{O}_{\mathsf{sk}}, \mathcal{O}_{\mathsf{eq}}}(1^\lambda, p)$
$\mathcal{O}_{\mathsf{chal}}\left((\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}), (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)})\right)$
$\beta' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{add}}, \mathcal{O}_{\mathsf{pair}}, \mathcal{O}_{\mathsf{sk}}, \mathcal{O}_{\mathsf{eq}}}(1^\lambda, p)$
If $\beta = \beta'$, and for all $f \in Q_{\mathsf{sk}}$, $f(\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}) = f(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)})$, output 1. Otherwise, output 0.

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

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

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

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

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

**Fig. 6.** Experiment $\mathsf{Exp}_2$. Wlog. we assume no query contains indices $i, j \in \mathbb{N}$ that exceed the size of the involved lists.

Definition 14]. Thus, we can use their master theorem [BCFG17, 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 [BCFG17, Theorem 7]) states that:

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

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

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

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

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

where the equality is taken in the ring $\mathbb{Z}_p[\boldsymbol{S}, \boldsymbol{T}, A, B, C, D, \Gamma]$, 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 $\mathsf{pk}$, the challenge ciphertext, and functional decryption keys, does not break the security of the scheme. We prove this condition is satisfied in Lemma 3 below.

**Lemma 3 (Symbolic Security).** *For any $(\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}), (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}) \in Z_p^{2n}$, and any set $Q_{\mathsf{sk}} \subseteq \mathcal{F}_{n,B_x,B_y,B_f}$ such that for all $f \in Q_{\mathsf{sk}}$, $f(\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}) = f(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)})$, we have:*

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

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

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

*where the equality is taken in the ring* $\mathbb{Z}_p[\boldsymbol{S},\boldsymbol{T},A,B,C,D,\Gamma]$*, 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 $\mathsf{Eq}_b(\mathbf{M})$. We prove it also satisfies $\mathsf{Eq}_{1-b}(\mathbf{M})$. To do so, we use the following rules:

**Rule 1** : for all $P,Q,R \in \mathbb{Z}_p[\boldsymbol{S},\boldsymbol{T},A,B,C,D,\Gamma]$, 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 $P \in \mathbb{Z}_p[\boldsymbol{S},\boldsymbol{T},A,B,C,D,\Gamma]$, any variable $X$ among the set $\{\boldsymbol{S},\boldsymbol{T},A,B,C,D,\Gamma\}$, 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 $\mathsf{Eq}_b(\mathbf{M})$ on $B = D = 0$ (using **Rule 2**), then using **Rule 1** on $P = C\Gamma S_i T_j$ for all $i,j \in [n]$, we obtain that:

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

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:

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

Thus, we have:

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

Using **Rule 1** on $P = (AD-BC)S_i B T_j$ for all $i,j \in [n]$ in the equation $\mathsf{Eq}_b(\mathbf{M})$, 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:

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

We write:

$$\begin{pmatrix} 0 \\ \mathbf{0} \\ 0 \\ D\boldsymbol{x}^{(b)} - \Gamma C\boldsymbol{S} \\ -B\boldsymbol{x}^{(b)} + \Gamma A\boldsymbol{S} \end{pmatrix}^{\top} \mathbf{M} \begin{pmatrix} 0 \\ \mathbf{0} \\ A\boldsymbol{y}^{(b)} - B\boldsymbol{T} \\ C\boldsymbol{y}^{(b)} - D\boldsymbol{T} \\ \mathbf{0} \end{pmatrix}$$
$$= \sum_{i,j\in[n]} \begin{pmatrix} Dx_i^{(b)} - \Gamma CS_i \\ -Bx_i^{(b)} + \Gamma AS_i \end{pmatrix}^{\top}$$
$$\times \left( m_{i,j}^{(1)} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + m_{i,j}^{(2)} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + m_{i,j}^{(3)} \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} + m_{i,j}^{(4)} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \right)$$
$$\times \begin{pmatrix} Ay_j^{(b)} - BT_j \\ Cy_j^{(b)} - DT_j \end{pmatrix}$$

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

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

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

Putting everything together, can write

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

as

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

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

Evaluating equation $\mathsf{Eq}_b(\mathbf{M})$ 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 $\mathsf{Eq}_b(\mathbf{M})$ 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:

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

Evaluating equation $\mathsf{Eq}_b(\mathbf{M})$ 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 $\mathsf{Eq}_b(\mathbf{M})$ 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:

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

Overall, we have:

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

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

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

and then, under (3)

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

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

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