Decentralized Evaluation of Quadratic Polynomials on Encrypted Data

Chloé Hébant[1,2], Duong Hieu Phan[3], and David Pointcheval[1,2]

[1] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
[2] INRIA, Paris, France
[3] Université de Limoges, France

Abstract Machine learning and group testing are quite useful methods for many different fields such as finance, banks, biology, medicine, etc. These application domains use quite sensitive data, and huge amounts of data. As a consequence, one would like to be able to both privately and efficiently compute on big data. While fully homomorphic encryption can be seen as a very powerful tool for such a task, it might not be efficient enough, and namely because of the very large ciphertexts. In addition, the result being encrypted, efficient distributed decryption is important to control who can get the information.

For our applications, we first remark that 2-DNF formulae evaluation is enough, but efficient multi-party decryption is still required to guarantee privacy. Boneh-Goh-Nissim proposed a nice encryption scheme that supports additions, one multiplication layer, and additions, by using a bilinear map on a composite-order group: this is perfectly suited for 2-DNF formulae evaluation. However, computations on such elliptic curves with composite order turned out to be quite inefficient, and namely when multi-party decryption is required. Fortunately, Freeman proposed a generalization, based on prime-order groups, with the same properties, but better efficiency.

Whereas the BGN cryptosystem relies on integer factoring for the trapdoor in the composite-order group, and thus possesses one public/secret key only, our first contribution is to show how the Freeman cryptosystem can handle multiple users with one general setup that just needs to define a pairing-based algebraic structure. Users' keys are efficient to generate and can also support efficient multi-party decryption, without a trusted server, hence in a fully decentralized setting. Fortunately, it helps to efficiently address some machine learning models and the group testing on encrypted data, without central authority.

1 Introduction

Decentralized Cryptography is one of the main directions of research in cryptography, especially in a concurrent environment of multi-user applications, where there is no need to trust any authority. Recently, the rise of blockchain's applications also witnessed the importance of decentralized applications. However, the blockchain mainly addresses the decentralized validation of transactions, but it does not help to decentralize the computations, which is the main goal of cryptographic protocols. For the computational purpose, though general solutions can be achieved via multi-party computation, reasonably efficient solutions only exist for a limited number of protocols, as decentralization usually adds constraints to the design of protocols: in broadcast encryption [FN94], the decentralized protocol in [PPS12] is much less efficient than the underlying original protocol [NNL01]; in attribute-based encryption [SW05], the decentralized scheme [LW11] requires bilinear groups of composite order; etc.

Decentralized Computing over Encrypted Data. In the last decade, the most active research direction carries on computing over encrypted data, with the seminal papers on Fully Homomorphic Encryption (FHE) [Gen09, BGN05] and on Functional Encryption (FE) [SW05, BSW11, GKP$^+$13, GGH$^+$13]. FE was generalized to the case of multi-user setting via the notion of multi-input/multi-client FE[GGG$^+$14, GGJS13, GKL$^+$13]. In this setting, it was a natural problem to consider the decentralization, which was recently addressed by Chotard *et al.* [CDG$^+$17], where each client encrypts his own input, all the clients agree and contribute to generate the functional decryption keys, there is no need of central authority anymore. Remark that, in functional encryption, there are efficient solutions for quadratic functions [Gay16, BCFG17, DGP18] but actually, only linear function evaluations can be decentralized. As far as we know, this is actually

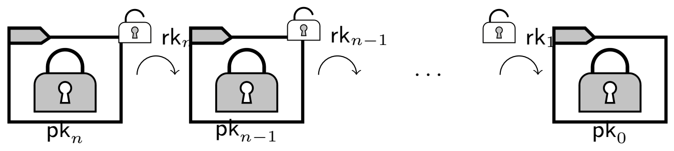

Figure 1. Distributed decryption

the only work done on decentralizing the computation over encrypted data. We consider in this paper the practical case of decentralized FHE in multi-user setting. Because the general solution for FHE is still not yet practical, we consider the real-life applications which only require the evaluation of quadratic polynomials.

1.1 Technical Contribution

We design an efficient distributed evaluation for quadratic polynomials, with decentralized generation of the keys. Boneh-Goh-Nissim [BGN05] proposed a nice solution for quadratic polynomials evaluation. However, their solution relies on a composite-order elliptic curve and thus on the hardness of the integer factoring. This possibly leads to a distributed solution, but that is highly inefficient. In fact, no efficient multi-party generation of distributed RSA modulus is known, except to 2 parties. But even the recent construction [FLOP18], the most efficient up to now, is still quite inefficient as it relies on oblivious transfer in the semi-honest setting, and on an IND-CPA encryption scheme, coin-tossing, zero-knowledge and secure two-party computation protocols in the malicious setting. Catalano and Fiore [CF15] introduced an efficient technique to transform a linearly-homomorphic encryption into a scheme able to evaluate quadratic operations on ciphertexts. However, they only consider a subclass of degree-2 polynomials where the number of additions of degree-2 terms is bounded by a constant. This is not enough for our applications and we do not try to decentralize this protocol.

Our Approach. Freeman [Fre10] proposed a conversion from composite-order groups to prime-order groups for the purpose of improving the efficiency. Interestingly, we show that the Freeman's conversion is well-suited for decentralized evaluation of 2-DNF formulae. In fact, by working in prime-order groups, we can avoid the difficulty of a distributed generation of RSA moduli. However, this is not enough to achieve a distributed scheme, with distributed decryption.

The idea behind our technique is that, from a k-somewhat homomorphic encryption (that allows k multiplicative-depth) and with the possibility of realizing a kind of proxy re-encryption, one can construct an efficient distributed evaluation for k-DNF formula. More precisely, thanks to the k-somewhat homomorphism, the k-DNF formula can be evaluated over ciphertexts. However, this should be done with an encryption scheme that allows distributed decryption, under keys that can be generated without any trusted party. This is where proxy re-encryption helps: each user i owns a "share" rk_i that is a proxy re-encryption key for a ciphertext under some public key pk_i into a ciphertext under the public key pk_{i-1}. Hence, if the computations are performed under pk_n, the users n to 1 can successively re-encrypt under $\mathsf{pk}_{n-1}, \dots,$ until pk_0, for which the secret key is publicly known, as shown in Figure 1.1. With a FHE scheme, one can perform this kind of proxy re-encryption, using an encryption of sk_n under pk_{n-1}, and thus obtain a distributed evaluation for any DNF formula. However, the current implementations of FHE are still far from efficient. While the BGN scheme does not seem to efficiently support the proxy re-encryption, we show that its variant via Freeman's conversion supports proxy re-encryption and thus leads to an efficient distributed evaluation for 2-DNF formula.

An open problem remains to construct an efficient distributed evaluation for k-DNF formulae. At first glance, one might think that a k-somewhat homomorphic encryption scheme can be transformed into a decentralized scheme by just adding a secret sharing on the top of the key generation. However, the current methods of secret sharing require a dealer during the setup, which is not compatible with our decentralized setting. In the particular case of BGN, no efficient distributed key generation is known. And so, an interesting direction is to follow our approach to consider an efficient k-somewhat homomorphic encryption scheme and add a proxy re-encryption technique.

1.2 Applications

Boneh, Goh, and Nissim proposed two main applications to secure evaluation of quadratic polynomials: private information retrieval schemes (PIR) and electronic voting protocols. We propose two more applications that are related to the group testing and the consistency model in machine learning. Our applications are particularly useful in practice in a decentralized setting, as they deal with sensitive data. Interestingly, the use of distributed evaluation for quadratic polynomials in this application is highly non-trivial and will be explained in the section on applications.

2 Generalities

2.1 Notations

We denote by $x \overset{\$}{\leftarrow} X$ the process of selecting x uniformly at random in the set X. The vectors $\mathbf{x} = (x_i)_i$ and matrices $\mathbf{M} = (m_{i,j})_{ij}$ are in bold and the vectors are written as row vectors, with sometimes components separated by commas for clarity: if $\mathbf{x} \overset{\$}{\leftarrow} X^n$, $\mathbf{x} = (x_1\ x_2\ \cdots\ x_n) = (x_1, x_2, \cdots, x_n)$.

Let $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ be the ring of integers *modulo p*, we denote by $\mathcal{M}_{m,n}(\mathbb{Z}_p)$ the set of matrices on $\mathbb{Z}_p$, of size $m \times n$, and thus m row-vectors of length n. $(\mathcal{M}_{m,n}(\mathbb{Z}_p), +)$ is an Abelian group. When $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p)$ and $\mathbf{B} \in \mathcal{M}_{n,n'}(\mathbb{Z}_p)$, the matrix product is denoted $\mathbf{A} \times \mathbf{B} \in \mathcal{M}_{m,n'}(\mathbb{Z}_p)$, or just $\mathbf{AB}$ if there is no ambiguity. $(\mathcal{M}_{n,n}(\mathbb{Z}_p), +, \times)$ is a ring, and we denote by $\mathrm{GL}_n(\mathbb{Z}_p) \subset \mathcal{M}_{n,n}(\mathbb{Z}_p) = \mathcal{M}_n(\mathbb{Z}_p)$ the subset of the invertible matrices of size n (for the above matrix product $\times$), which is also called the *general linear group*. We denote by $\mathrm{SL}_n(\mathbb{Z}_p) \subset \mathrm{GL}_n(\mathbb{Z}_p)$ the subset of the invertible matrices of determinant 1, which is also called the *special linear group*.

We will use the tensor product: for two vectors $\mathbf{a} = (a_1, a_2, \cdots, a_n) \in \mathbb{Z}_p^n$ and $\mathbf{b} = (b_1, b_2, \cdots, b_m) \in \mathbb{Z}_p^m$, the tensor product $\mathbf{a} \otimes \mathbf{b}$ is the vector $(a_1\mathbf{b}, \cdots, a_n\mathbf{b}) = (a_1b_1, \cdots, a_1b_m, a_2b_1, \cdots, a_2b_m, \cdots, a_nb_m) \in \mathbb{Z}_p^{mn}$; and for two matrices $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p)$ and $\mathbf{B} \in \mathcal{M}_{m',n'}(\mathbb{Z}_p)$,

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_m \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_{m'} \end{pmatrix} \quad \mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 \\ \mathbf{a}_1 \otimes \mathbf{b}_2 \\ \vdots \\ \mathbf{a}_m \otimes \mathbf{b}_{m'} \end{pmatrix} \in \mathcal{M}_{mm',nn'}(\mathbb{Z}_p).$$

Because of the bi-linearity of the tensor product, we have, for $\mathbf{A}, \mathbf{A}' \in \mathcal{M}_{m,n}(\mathbb{Z}_p)$ and $\mathbf{B}, \mathbf{B}' \in \mathcal{M}_{m',n'}(\mathbb{Z}_p)$,

$$(\mathbf{A} + \mathbf{A}') \otimes (\mathbf{B} + \mathbf{B}') = (\mathbf{A} \otimes \mathbf{B}) + (\mathbf{A} \otimes \mathbf{B}') + (\mathbf{A} \otimes \mathbf{B}') + (\mathbf{A}' \otimes \mathbf{B}')$$

We will also use the following important relation between matrix product and tensor product: for $\mathbf{A} \in \mathcal{M}_{m,k}(\mathbb{Z}_p)$, $\mathbf{A}' \in \mathcal{M}_{k,n}(\mathbb{Z}_p)$, $\mathbf{B} \in \mathcal{M}_{m',k'}(\mathbb{Z}_p)$, and $\mathbf{B}' \in \mathcal{M}_{k',n'}(\mathbb{Z}_p)$, $(\mathbf{A} \times \mathbf{A}') \otimes (\mathbf{B} \times \mathbf{B}') = (\mathbf{A} \otimes \mathbf{B}) \times (\mathbf{A}' \otimes \mathbf{B}')$.

For any group $\mathbb{G}$, we denote by $\langle g \rangle$ the space generated by $g \in \mathbb{G}$. For a generator g of a cyclic group $\mathbb{G} = \langle g \rangle$, we use the implicit representation $[a]$ of any element $h = g^a \in \mathbb{G}$ and by extension we will use the "bracket" notations, which makes use of the above matrix properties over the exponents, that are scalars in $\mathbb{Z}_p$ when $\mathbb{G}$ is a cyclic group of order p. See Figure 2 for more notations with "brackets".

$$\begin{array}{rccl}
a \in \mathbb{Z}_p, & [a] & = & g^a \\
\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p), & [\mathbf{A}] & = & g^{\mathbf{A}} = (g^{a_{ij}})_{ij} \\
x \in \mathbb{Z}_p, \mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p), & x \cdot [\mathbf{A}] & = & g^{x\mathbf{A}} = ((g^{a_{ij}})^x)_{ij} \\
\mathbf{X} \in \mathcal{M}_{n,m'}(\mathbb{Z}_p), \mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p), & [\mathbf{A}] \cdot \mathbf{X} & = & g^{\mathbf{AX}} = \left(\prod_{k=1}^{m} (g^{a_{ik}})^{x_{kj}}\right)_{ij} \\
\mathbf{X} \in \mathcal{M}_{n',m}(\mathbb{Z}_p), \mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p), & \mathbf{X} \cdot [\mathbf{A}] & = & g^{\mathbf{XA}} = \left(\prod_{k=1}^{m} (g^{a_{kj}})^{x_{ik}}\right)_{ij} \\
\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p), \mathbf{B} \in \mathcal{M}_{m,n}(\mathbb{Z}_p), & \begin{array}{c}[\mathbf{A}] \boldsymbol{+} [\mathbf{B}] \\ = \\ [\mathbf{A}+\mathbf{B}]\end{array} & = & g^{\mathbf{A}+\mathbf{B}} = \left(g^{a_{ij}} \cdot g^{b_{ij}}\right)_{ij}
\end{array}$$

Figure 2. Bracket notations

2.2 Projections

In order to continue with matrix properties and linear applications, a *projection* π is a linear function such that $\pi \circ \pi = \pi$. When the image is of dimension one, any projection can be represented by the matrix

$$\mathbf{U}_n = \begin{pmatrix} 0 \dots 0\,0 \\ 0 \ddots 0\,0 \\ 0 \dots 0\,0 \\ 0 \dots 0\,1 \end{pmatrix}$$

which is the canonical projection onto the 1-dimension image $\langle(0,0,\dots,0,1)\rangle$ along the kernel $\langle(1,0,\dots,0,0),(0,1,\dots,0,0),\dots,(0,0,\dots,1,0)\rangle$, in an appropriate basis $\mathcal{B}$. Indeed, the projection along the subspace $K = \langle \mathbf{p}_1,\dots,\mathbf{p}_{n-1}\rangle$ onto $\langle \mathbf{b} \rangle$ is represented by the matrix $\mathbf{P} = \mathbf{B}^{-1}\mathbf{U}_n\mathbf{B}$ where $\mathbf{B}$ is the change of basis matrix, from the canonical basis to the basis $\mathcal{B} = \{\mathbf{b}_1,\dots,\mathbf{b}_{n-1},\mathbf{p}\}$. This is actually

$$\mathbf{B} = \begin{pmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_{n-1} \\ \mathbf{b} \end{pmatrix}.$$

And for any projection π, with representation matrix $\mathbf{P}$, there exists a unique change of basis matrix $\mathbf{B} \in \mathrm{SL}_n(\mathbb{Z}_p)$. We will call it the change of basis matrix associated to $\mathbf{P}$. From this matrix $\mathbf{B}$, we can parse as above with vectors $\mathbf{p}_1, \dots, \mathbf{p}_{n-1}$ and $\mathbf{b}$, the set $\{\mathbf{p}_1,\dots,\mathbf{p}_{n-1}\}$ is a basis of the kernel K and $\langle \mathbf{b} \rangle$ is the image.

Given two projections π_1 and π_2 in $\mathbb{Z}_p^2$, they are represented by $\mathbf{P}_1 = \mathbf{B}_1^{-1}\mathbf{U}_n\mathbf{B}_1$ and $\mathbf{P}_2 = \mathbf{B}_2^{-1}\mathbf{U}_n\mathbf{B}_2$, respectively, the tensor product $\pi = \pi_1 \otimes \pi_2$ is represented by $\mathbf{P} = \mathbf{P}_1 \otimes \mathbf{P}_2$, that is equal to

$$\begin{aligned}(\mathbf{B}_1^{-1}\mathbf{U}_n\mathbf{B}_1) \otimes (\mathbf{B}_2^{-1}\mathbf{U}_n\mathbf{B}_2) &= (\mathbf{B}_1^{-1} \otimes \mathbf{B}_2^{-1}) \times (\mathbf{U}_n \otimes \mathbf{U}_n) \times (\mathbf{B}_1 \otimes \mathbf{B}_2) \\ &= (\mathbf{B}_1 \otimes \mathbf{B}_2)^{-1} \times \mathbf{U}_{n^2} \times (\mathbf{B}_1 \otimes \mathbf{B}_2).\end{aligned}$$

This is thus also a projection in $\mathbb{Z}_p^4$ with image of dimension 1, and the associated change of basis matrix is $\mathbf{B} = \mathbf{B}_1 \otimes \mathbf{B}_2$. In the particular case of dimension 2,

$$\mathbf{B}_1 = \begin{pmatrix} \mathbf{p}_1 \\ \mathbf{b}_1 \end{pmatrix} \text{ and } \mathbf{B}_2 = \begin{pmatrix} \mathbf{p}_2 \\ \mathbf{b}_2 \end{pmatrix}, \quad \text{then } \mathbf{B} = \mathbf{B}_1 \otimes \mathbf{B}_2 = \begin{pmatrix} \mathbf{p}_1 \otimes \mathbf{p}_2 \\ \mathbf{p}_1 \otimes \mathbf{b}_2 \\ \mathbf{b}_1 \otimes \mathbf{p}_2 \\ \mathbf{b}_1 \otimes \mathbf{b}_2 \end{pmatrix}.$$

Hence, the image of $\pi = \pi_1 \otimes \pi_2$ is spanned by $\mathbf{b} = \mathbf{b}_1 \otimes \mathbf{b}_2$, while $\{\mathbf{p}_1 \otimes \mathbf{p}_2, \mathbf{p}_1 \otimes \mathbf{b}_2, \mathbf{b}_1 \otimes \mathbf{p}_2\}$ is a basis of the kernel:

$$\begin{aligned}\ker(\pi) &= \langle \mathbf{p}_1 \otimes \mathbf{p}_2, \mathbf{p}_1 \otimes \mathbf{b}_2, \mathbf{b}_1 \otimes \mathbf{p}_2 \rangle \\ &= \{x \cdot \mathbf{p}_1 \otimes \mathbf{p}_2 + y_2 \cdot \mathbf{p}_1 \otimes \mathbf{b}_2 + y_1 \cdot \mathbf{b}_1 \otimes \mathbf{p}_2, x, y_1, y_2 \in \mathbb{Z}_p\} \\ &= \{(x_1 + x_2) \cdot \mathbf{p}_1 \otimes \mathbf{p}_2 + y_2 \cdot \mathbf{p}_1 \otimes \mathbf{b}_2 + y_1 \cdot \mathbf{b}_1 \otimes \mathbf{p}_2, x_1, x_2, y_1, y_2 \in \mathbb{Z}_p\} \\ &= \{\mathbf{p}_1 \otimes (x_2 \cdot \mathbf{p}_2 + y_2 \cdot \mathbf{b}_2) + (x_1 \cdot \mathbf{p}_1 + y_1 \cdot \mathbf{b}_1) \otimes \mathbf{p}_2, x_1, x_2, y_1, y_2 \in \mathbb{Z}_p\}\end{aligned}$$

As a consequence, $\mathbf{p}_1 \otimes \mathbf{r}_2 + \mathbf{r}_1 \otimes \mathbf{p}_2$, for $\mathbf{r}_1, \mathbf{r}_2 \overset{\$}{\leftarrow} \mathbb{Z}_p^2$, provides a uniform sampling in $\ker(\pi)$.

2.3 Bilinear Group Generator

A *bilinear group generator* $\mathcal{G}$ is an algorithm that takes as input a security parameter λ and outputs a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ such that $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ are cyclic groups of prime order p (a λ-bit prime integer), and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an *admissible pairing*:

- e is bilinear: for all $a, b \in \mathbb{Z}_p$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$;
- e is efficiently computable (in polynomial-time in λ);
- e is non-degenerate: $e(g_1, g_2) \neq 1$.

Furthermore, the bilinear setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ is said *asymmetric* when $\mathbb{G}_1 \neq \mathbb{G}_2$.

In such a case, we have different groups $\mathbb{G}_s$, for $s = 1, 2, T$, so we will use $[\cdot]_s$ to specify to which group the "bracket" representation refers. Since we now have another law with the pairing operation, we will use the notation $[a]_1 \bullet [b]_2 = [a \cdot b]_T$ for $[a]_1 \in \mathbb{G}_1, [b]_2 \in \mathbb{G}_2$. We also define, for $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p)$ and $\mathbf{B} \in \mathcal{M}_{m',n'}(\mathbb{Z}_p)$, $[\mathbf{A}]_1 \bullet [\mathbf{B}]_2 = [\mathbf{A} \otimes \mathbf{B}]_T$, which can be evaluated with pairing operations between $\mathbb{G}_1$ and $\mathbb{G}_2$ group elements.

3 Encryption for Evaluation of 2-DNF Formulae

3.1 BGN and Freeman Cryptosystems

To evaluate 2-DNF formulae on encrypted data, Boneh-Goh-Nissim described a cryptosystem [BGN05] that supports additions, one multiplication layer, and additions. They used a bilinear map on a composite-order group and the secret key is the factorization of the order of the group. We recall it bellow:

Keygen(λ): Given a security parameter $\lambda \in \mathbb{Z}^+$, it generates a symmetric bilinear setting, with two groups $\mathbb{G}, \mathbb{G}_T$, of composite order $n = pq$, and a pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Let $\mathbb{G}_1$ be the subgroup of $\mathbb{G}$ of order p. It picks two random generators $g, u \overset{\$}{\leftarrow} \mathbb{G}$ and sets $h = u^q$ (a generator of $\mathbb{G}_1$). The public key is $\mathsf{pk} = ((\mathbb{G}, \mathbb{G}_T, n, g, e), \mathbb{G}_1, h)$ and the private key is $\mathsf{sk} = p$.

Encrypt(pk, m): To encrypt a message m using public key pk, it picks a random $r \overset{\$}{\leftarrow} \mathbb{Z}_n$ and outputs the ciphertext $C = g^m h^r \in \mathbb{G}$.

Decrypt(sk, C): To decrypt a ciphertext C using the private key sk, one outputs $m \leftarrow \log_{g^{\mathsf{sk}}}(C^{\mathsf{sk}})$.

The intuition behind this construction is the random noise h^r in $\mathbb{G}_1$, that hides g^m during the encryption, and that can be canceled during the decryption using the private order p. This encryption scheme is clearly additively homomorphic in $\mathbb{Z}_q$, which allows additions. By applying the pairing operation between two ciphertexts, one gets a new ciphertext of the product of the plaintexts (still in $\mathbb{Z}_q$), but now under a new similar encryption scheme in $\mathbb{G}_T$. Additions are again possible, hence the evaluation of 2-DNF formulae. Unfortunately, composite-order groups require huge orders, since the factorization must me difficult, with costly pairing evaluations.

In addition, one has to compute a discrete logarithm for the decryption process, which limits to small plaintexts.

In order to improve on the efficiency, and address the first above issue, Freeman in [Fre10, Section 5] proposed a system on prime-order groups, using a similar property of noise that can be removed, with the general definition of subgroup decision problem. Let us recall the Freeman's cryptosystem:

$\mathsf{Keygen}(\lambda)$**:** Given a security parameter $\lambda \in \mathbb{Z}^+$, it generates a description of three Abelian groups G, H, G_T and a pairing $e : G \times H \to G_T$. It also generates a description of two subgroups $G_1 \subset G, H_1 \subset H$ and two homomorphisms π_1, π_2 such that G_1, H_1 are contained in the kernels of π_1, π_2 respectively. It picks $g \xleftarrow{\$} G$ and $h \xleftarrow{\$} H$, and outputs the public key $\mathsf{pk} = (G, H, g, h, G_1, H_1)$ and the private key $\mathsf{sk} = (\pi_1, \pi_2)$.

$\mathsf{Encrypt}(\mathsf{pk}, m)$**:** To encrypt a message m using public key pk, one picks $g_1 \xleftarrow{\$} G_1$ and $h_1 \xleftarrow{\$} H_1$, and outputs the ciphertext $(C_A, C_B) = (g^m \cdot g_1, h^m \cdot h_1) \in G \times H$.

$\mathsf{Decrypt}(\mathsf{sk}, C)$**:** Given $C = (C_A, C_B)$, output $m \leftarrow \log_{\pi_1(g)}(\pi_1(C_A))$ (which should be the same as $\log_{\pi_2(h)}(\pi_2(C_B))$).

The Freeman's scheme is also additively homomorphic. Moroever, if an homomorphism π_T exists such that, for all $g \in G, h \in H$, $e(\pi_1(g), \pi_2(h)) = \pi_T(e(g, h))$, we can get, as above, a ciphertext in G_T of the product of the two plaintexts, when multiplying the ciphertexts in G and H. The new encryption scheme in G_T is still additively homomorphic, and allows evaluations of 2-DNF formulae.

Remark 1. We note that in the Freeman's cryptosystem, ciphertexts contain encryptions of m in both G and H to allow any kind of additions and multiplication. But one could focus on just one ciphertext when one knows the formula to be evaluated.

3.2 Our Cryptosystem

The main goal of Freeman's approach was to generalize the BGN cryptosystem to any hard-subgroup problems, which allows applications to prime-order groups under the classical Decisional Diffie-Hellman or Decisional Linear assumptions, with high gain in efficiency. But we insist in this paper on a quite interesting property: while the bilinear setting in the BGN construction is specific for one user, because of the private decryption key that contains the factorization, the Freeman's approach can apply to multi-user settings.

To this aim, we show that we can split the Freeman's Keygen algorithm into separate Setup and Keygen algorithms. Indeed, for concrete instantiation under the Decisional Diffie-Hellman problem, the private key is a projection. And one can generate many independent projections for different users.

We now present our variant of the Freeman's cryptosystem supporting multiple users, with our above notations and without the twin ciphertexts (in G and H). Since we will work in groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, the algorithms Keygen, Encrypt and Decrypt will take a sub-script s to precise the group $\mathbb{G}_s$ in which they operate.

$\mathsf{Setup}(\lambda)$**:** Given a security parameter $\lambda \in \mathbb{N}$, run and output

$$\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathcal{G}(\lambda).$$

$\mathsf{Keygen}_s(\mathsf{param})$**:** For $s \in \{1, 2\}$. Choose $\mathbf{B}_s \xleftarrow{\$} \mathrm{SL}_2(\mathbb{Z}_p)$. Let $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}_2\mathbf{B}_s$ and $\mathbf{p}_s \in \ker(\mathbf{P}_s) \setminus \{\mathbf{0}\}$. Output the public key $\mathsf{pk}_s = [\mathbf{p}_s]_s$ and the private key $\mathsf{sk}_s = \mathbf{P}_s$. In the following, we always implicitly assume that the public keys contain the public parameters param, and the private keys contain the public keys.

From $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Keygen}_1(\mathsf{param})$ and $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{Keygen}_2(\mathsf{param})$, one can consider $\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathsf{sk}_T = (\mathsf{sk}_1, \mathsf{sk}_2)$, which are associated public and private keys in $\mathbb{G}_T$, as we explain below.

$\mathsf{Encrypt}_s(\mathsf{pk}_s, m, A_s)$**:** For $s \in \{1,2\}$, to encrypt a message m using public key pk_s and $A_s = [\mathbf{a}]_s \in \mathbb{G}_s^2$, choose $r \overset{\$}{\leftarrow} \mathbb{Z}_p$ and output the ciphertext $C_s = (m \cdot [\mathbf{a}]_s + r \cdot [\mathbf{p}_s]_s, [\mathbf{a}]_s) \in \mathbb{G}_s^2 \times \mathbb{G}_s^2$.
For $s = T$, with $A_s = ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2)$, set $[\mathbf{a}]_T = [\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2 \in \mathbb{G}_T^4$, choose $[\mathbf{r}_1]_1 \overset{\$}{\leftarrow} \mathbb{G}_1^2, [\mathbf{r}_2]_2 \overset{\$}{\leftarrow} \mathbb{G}_2^2$, and output $C_T = (m \cdot [\mathbf{a}]_T + [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 + [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2, [\mathbf{a}]_T) \in \mathbb{G}_T^4 \times \mathbb{G}_T^4$.

$\mathsf{Decrypt}_s(\mathsf{sk}_s, C_s)$**:** For $s \in \{1,2\}$, given $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$ and $\mathsf{sk}_s = \mathbf{P}_s$, let $C'_s = ([\mathbf{c}_{s,1}]_s \cdot \mathbf{P}_s, [\mathbf{c}_{s,2}]_s \cdot \mathbf{P}_s)$. For $s = T$, compute $C'_T = ([\mathbf{c}_{T,1}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2), [\mathbf{c}_{T,2}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2))$. In both cases, output the logarithm of the first component of $\mathbf{c}'_{s,1}$ in base the first component of $\mathbf{c}'_{s,2}$.

As already explained above, the encryption process mask the message by an element in the kernel of a certain projection. The secret key is the corresponding projection $\mathbf{P}_s$ which later remove the mask by diving it into the kernel. In the $\mathsf{Decrypt}$ algorithm, C'_s is a Diffie-Hellman tuple (whatever the group under consideration), the discrete logarithm of one component is sufficient to decrypt, since the plaintext is the common exponent.

One can note that matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ are drawn independently, so the keys in $\mathbb{G}_1$ and $\mathbb{G}_2$ are independent. For any pair of keys $(\mathsf{pk}_1 = [\mathbf{p}_1]_1, \mathsf{pk}_2 = [\mathbf{p}_2]_2)$, one can implicitly define a public key for the target group. To decrypt in the target group, both private keys $\mathsf{sk}_1 = \mathbf{P}_1$ and $\mathsf{sk}_2 = \mathbf{P}_2$ are needed. Actually, one just needs $\mathbf{P}_1 \otimes \mathbf{P}_2$ to decrypt: $C'_T = ([\mathbf{c}_{T,1}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2), [\mathbf{c}_{T,2}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2))$, but $\mathbf{P}_1 \otimes \mathbf{P}_2$ and $(\mathbf{P}_1, \mathbf{P}_2)$ contain the same information and the latter is more compact.

With the algorithms defined above, we have three encryption schemes $\mathcal{E}_s$: ($\mathsf{Setup}, \mathsf{Keygen}_s$, $\mathsf{Encrypt}_s, \mathsf{Decrypt}_s$) for $s = 1, 2$ or T, with a common Setup. Let us study their correctness and their security properties.

Proposition 2. *For $s \in \{1,2,T\}$, $\mathcal{E}_s$ is correct.*

Proof. For $s = 1, 2$:

$$\begin{aligned}
[\mathbf{c}_{s,2}]_s \cdot \mathbf{P}_s &= [\mathbf{a}]_s \cdot \mathbf{P}_s = [\mathbf{a}\mathbf{P}_s]_s \\
[\mathbf{c}_{s,1}]_s \cdot \mathbf{P}_s &= (m \cdot [\mathbf{a}]_s + r \cdot [\mathbf{p}_s]_s) \cdot \mathbf{P}_s = m \cdot [\mathbf{a}]_s \cdot \mathbf{P}_s + r \cdot [\mathbf{p}_s]_s \cdot \mathbf{P}_s \\
&= m \cdot [\mathbf{a}\mathbf{P}_s]_s + r \cdot [\mathbf{p}_s\mathbf{P}_s]_s = m \cdot [\mathbf{a}\mathbf{P}_s]_s + r \cdot [\mathbf{0}]_s = m \cdot [\mathbf{a}\mathbf{P}_s]_s
\end{aligned}$$

For $s = T$:

$$\begin{aligned}
&[\mathbf{c}_{T,2}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) = [\mathbf{a}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) = [\mathbf{a}(\mathbf{P}_1 \otimes \mathbf{P}_2)]_T \\
&[\mathbf{c}_{T,1}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) = (m \cdot [\mathbf{a}]_T + [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 + [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2) \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) \\
&= m \cdot [\mathbf{a}(\mathbf{P}_1 \otimes \mathbf{P}_2)]_T + ([\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2) \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) + ([\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2) \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) \\
&= m \cdot [\mathbf{a}(\mathbf{P}_1 \otimes \mathbf{P}_2)]_T + [\mathbf{p}_1 \otimes \mathbf{r}_2]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) + [\mathbf{r}_1 \otimes \mathbf{p}_2]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) \\
&= m \cdot [\mathbf{a}(\mathbf{P}_1 \otimes \mathbf{P}_2)]_T + [\mathbf{p}_1\mathbf{P}_1 \otimes \mathbf{r}_2\mathbf{P}_2]_T + [\mathbf{r}_1\mathbf{P}_1 \otimes \mathbf{p}_2\mathbf{P}_2]_T \\
&= m \cdot [\mathbf{a}(\mathbf{P}_1 \otimes \mathbf{P}_2)]_T + [\mathbf{0} \otimes \mathbf{r}_2\mathbf{P}_2]_T + [\mathbf{r}_1\mathbf{P}_1 \otimes \mathbf{0}]_T = m \cdot [\mathbf{a}(\mathbf{P}_1 \otimes \mathbf{P}_2)]_T
\end{aligned}$$

In both cases, $C'_{s,1} = [\mathbf{c}'_{s,1}]_s = m \cdot [\mathbf{c}'_{s,2}]_s = m \cdot C'_{s,2}$. Whatever the size of the vectors, one discrete logarithm computation is enough to extract m.

3.3 Security Properties

In this section, we first recall the definition notion for encryption and the computational assumption we will rely on, and then we will state and prove the security results.

$$\begin{aligned}\mathrm{Exp}_{\mathcal{E}}^{\textsf{ind-cpa-}b}(\mathcal{A}):\ & \textsf{param} \leftarrow \textsf{Setup}(\lambda); (\textsf{sk}, \textsf{pk}) \leftarrow \textsf{Keygen}(\textsf{param}); m_0, m_1 \leftarrow \mathcal{A}(, \textsf{pk})\\ & C \leftarrow \textsf{Encrypt}_s(\textsf{pk}, m_b); b' \leftarrow \mathcal{A}(\textsf{pk}, C)\\ & \textbf{return } b'\end{aligned}$$

Figure 3. Experiment of IND-CPA

Definitions. We first recall the semantic security, *a.k.a.* indistinguishability (or IND-CPA), for a public-key encryption scheme, according to the experiment presented in Figure 3.

Definition 3. Let $\mathcal{E} = (\textsf{Setup}, \textsf{Keygen}, \textsf{Encrypt}, \textsf{Decrypt})$ be an encryption scheme. Let us denote $\mathrm{Exp}_{\mathcal{E}}^{\textsf{ind-cpa-}b}(\mathcal{A})$ the experiment defined in Figure 3. The advantage $\mathrm{Adv}_{\mathcal{E}}^{\textsf{ind-cpa}}(\mathcal{A})$ of an adversary $\mathcal{A}$ against *indistinguishability under chosen plaintext attacks* (IND-CPA) is defined by:

$$\Pr\left[\mathrm{Exp}_{\mathcal{E}}^{\textsf{ind-cpa-1}}(\mathcal{A}) = 1\right] - \Pr\left[\mathrm{Exp}_{\mathcal{E}}^{\textsf{ind-cpa-0}}(\mathcal{A}) = 1\right].$$

We say that an encryption scheme $\mathcal{E}$ is $(t, \varepsilon) - \textsf{IND-CPA}$ if for any adversary $\mathcal{A}$ running within time t, its advantage $\mathrm{Adv}_{\mathcal{E}}^{\textsf{ind-cpa}}(\mathcal{A})$ is bounded by ε.

We denote by $\mathrm{Adv}_{\mathcal{E}}^{\textsf{ind-cpa}}(t)$ the best advantage any adversary $\mathcal{A}$ can get within time t. Our security results will rely on the standard Decisional Diffie-Hellman assumption:

Definition 4. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p. The advantage $\mathrm{Adv}_{\mathbb{G}}^{\textsf{ddh}}(\mathcal{A})$ of an adversary $\mathcal{A}$ against the Decisional Diffie-Hellman (DDH) problem in $\mathbb{G}$ is defined by:

$$\Pr\left[\mathcal{A}(g, g^x, g^y, g^{xy}) = 1 | x, y \xleftarrow{\$} \mathbb{Z}_p\right] - \Pr\left[\mathcal{A}(g, g^x, g^y, g^z) = 1 | x, y, z \xleftarrow{\$} \mathbb{Z}_p\right].$$

We say that the DDH problem in $\mathbb{G}$ is (t, ε)-hard if for any advantage $\mathcal{A}$ running within time t, its advantage $\mathrm{Adv}_{\mathbb{G}}^{\textsf{ddh}}(\mathcal{A})$ is bounded by ε.

We denote by $\mathrm{Adv}_{\mathbb{G}}^{\textsf{ddh}}(t)$ the best advantage any adversary $\mathcal{A}$ can get within time t.

Theorem 5. *For $s \in \{1, 2\}$, $\mathcal{E}_s$ is* IND-CPA *under the* DDH *assumption in $\mathbb{G}_s$. More precisely, for any adversary $\mathcal{A}$ running within time t,*

$$\mathrm{Adv}_{\mathcal{E}_s}^{\textsf{ind-cpa}}(\mathcal{A}) \leq 2 \times \mathrm{Adv}_{\mathbb{G}_s}^{\textsf{ddh}}(t).$$

Proof. We denote by $\mathrm{Adv}_{s}^{\textsf{ind-cpa}}(\mathcal{A})$ the advantage of $\mathcal{A}$ against $\mathcal{E}_s$. We assume the running time of $\mathcal{A}$ is bounded by t.

Game $\mathbf{G}_0$: In this first game, the simulator plays the role of the challenger in the experiment $\mathrm{Exp}_{\mathcal{E}_s}^{\textsf{ind-cpa-0}}(\mathcal{A})$, where $b = 0$:
$\mathcal{S}(\lambda)$:
- $\textsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \textsf{Setup}(\lambda)$
- $(\textsf{sk}_s, \textsf{pk}_s) \leftarrow \textsf{Keygen}_s(\textsf{param})$
- $m_0, m_1, [\mathbf{a}]_s \leftarrow \mathcal{A}(\textsf{param}, \textsf{pk}_s)$
- $C_s = (m_0 \cdot [\mathbf{a}]_s + r \cdot [\mathbf{p}_s]_s, [\mathbf{a}]_s) \leftarrow \textsf{Encrypt}_s(\textsf{pk}_s, m_0, [\mathbf{a}]_s)$
- $b' \leftarrow \mathcal{A}(\textsf{param}, \textsf{pk}_s, C_s)$

We are interested in the event E: $b' = 1$. By definition,

$$\Pr_{\mathbf{G}_0}[E] = \Pr\left[\mathrm{Exp}_{\mathcal{E}}^{\textsf{ind-cpa-0}}(\mathcal{A}) = 1\right].$$

Game $\mathbf{G}_1$: Now the simulator takes as input a Diffie-Hellman tuple $([\mathbf{p}]_s, [\mathbf{r}]_s)$, with $\mathbf{r} = r \cdot \mathbf{p}$ for some scalar r, and emulates Keygen_s and $\mathsf{Encrypt}_s$ by defining $\mathsf{pk}_s \leftarrow [\mathbf{p}]_s$ and $C_s \leftarrow (m_0 \cdot [\mathbf{a}]_s + [\mathbf{r}]_s, [\mathbf{a}]_s)$. Thanks to the Diffie-Hellman tuple, this game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_1}[E] = \Pr_{\mathbf{G}_0}[E]$.

Game $\mathbf{G}_2$: The simulator now receives a random tuple $([\mathbf{p}]_s, [\mathbf{r}]_s)$: $\Pr_{\mathbf{G}_2}[E] - \Pr_{\mathbf{G}_1}[E] \leq \mathrm{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t)$.

Game $\mathbf{G}_3$: The simulator still receives a random tuple $([\mathbf{p}]_s, [\mathbf{r}]_s)$, but generates $C_s \leftarrow (m_1 \cdot [\mathbf{a}]_s + [\mathbf{r}]_s, [\mathbf{a}]_s)$. Thanks to the random mask $[\mathbf{r}]_s$, this game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_3}[E] = \Pr_{\mathbf{G}_2}[E]$.

Game $\mathbf{G}_4$: The simulator now receives a Diffie-Hellman tuple $([\mathbf{p}]_s, [\mathbf{r}]_s)$, with $\mathbf{r} = r \cdot \mathbf{p}$ for some scalar r: $\Pr_{\mathbf{G}_4}[E] - \Pr_{\mathbf{G}_3}[E] \leq \mathrm{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t)$.

Game $\mathbf{G}_5$: In this game, the simulator can perfectly emulate the challenger in the experiment $\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_s}(\mathcal{A})$, where $b = 1$: This game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_5}[E] = \Pr_{\mathbf{G}_4}[E]$.

One can note, that in this last game, we have $\Pr_{\mathbf{G}_5}[E] = \Pr\left[\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_s}(\mathcal{A}) = 1\right]$, hence

$$\begin{aligned}\Pr\left[\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_s}(\mathcal{A}) = 1\right] - \Pr\left[\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_s}(\mathcal{A}) = 1\right] &= \Pr_{\mathbf{G}_5}(E) - \Pr_{\mathbf{G}_0}(E) \\ &\leq 2 \times \mathrm{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t),\end{aligned}$$

which concludes the proof.

Corollary 6. *$\mathcal{E}_T$ is $\mathsf{IND\text{-}CPA}$ under the DDH assumption in either $\mathbb{G}_1$ or $\mathbb{G}_2$. More precisely, for any adversary $\mathcal{A}$ running within time t,*

$$\mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_T}(\mathcal{A}) \leq 2 \times \min\{\mathrm{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t + t_m + t_e), \mathrm{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t + t_m + t_e)\},$$

where t_m is the time for one multiplication and t_e the time for one encryption.

Proof. The semantic security for ciphertexts in $\mathbb{G}_T$ comes from the fact that:

$$\begin{aligned}&\mathsf{Encrypt}_T(\mathsf{pk}_T, m, ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2)) \\ &= \mathsf{Multiply}(\mathsf{Encrypt}_1(\mathsf{pk}_1, m, [\mathbf{a}_1]_1), \mathsf{Encrypt}_2(\mathsf{pk}_2, 1, [\mathbf{a}_2]_2)) \\ &= \mathsf{Multiply}(\mathsf{Encrypt}_1(\mathsf{pk}_1, 1, [\mathbf{a}_1]_1), \mathsf{Encrypt}_2(\mathsf{pk}_2, m, [\mathbf{a}_2]_2))\end{aligned}$$

Indeed, with this relation, each ciphertext in $\mathbb{G}_1$ can be transformed into a ciphertext in $\mathbb{G}_T$ (idem with a ciphertext in $\mathbb{G}_2$). Let $\mathcal{A}$ be an adversary against $\mathsf{IND\text{-}CPA}$ of $\mathcal{E}_T$, in $\mathbb{G}_T$.

Game $\mathbf{G}_0$: In the first game, the simulator plays the role of the challenger in the experiment $\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_T}(\mathcal{A})$, where $b = 0$:
$\mathcal{S}(\lambda)$:
- $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{Setup}(\lambda)$
- $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{Keygen}_1(\mathsf{param})$,$(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{Keygen}_2(\mathsf{param})$
- $m_0, m_1, [\mathbf{a}]_1, [\mathbf{a}]_2 \leftarrow \mathcal{A}(\mathsf{param}, (\mathsf{pk}_1, \mathsf{pk}_2))$
- $C_T = \mathsf{Encrypt}_T((\mathsf{pk}_1, \mathsf{pk}_2), m_0, ([\mathbf{a}]_1, [\mathbf{a}]_2))$
- $\beta \leftarrow \mathcal{A}(\mathsf{param}, (\mathsf{pk}_1, \mathsf{pk}_2), C_T)$

We are interested in the event E: $b' = 1$. By definition,

$$\Pr_{\mathbf{G}_0}[E] = \Pr\left[\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_T}(\mathcal{A}) = 1\right].$$

Game $\mathbf{G}_1$: The simulator interacts with a challenger in the experiment $\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_1}(\mathcal{A})$, where $b = 0$. It thus first receives $\mathsf{param}, \mathsf{pk}_1$ from that challenger, generates pk_2 by himself to provide $(\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2))$ to the adversary. The latter sends back $(m_0, m_1, [\mathbf{a}]_1, [\mathbf{a}]_2)$, from which it sends $(m_0, m_1, [\mathbf{a}]_1)$ to the challenger. It gets back $C_1 = \mathsf{Encrypt}_1(\mathsf{pk}_1, m_0, [\mathbf{a}]_1)$. It can compute $C_T = \mathsf{Multiply}(C_1, \mathsf{Encrypt}_2(\mathsf{pk}_2, 1, [\mathbf{a}_2]_2))$, to be sent to the adversary. This game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_1}[E] = \Pr_{\mathbf{G}_0}[E]$.

Game $\mathbf{G}_2$: The simulator interacts with a challenger in the experiment $\mathrm{Exp}_{\mathcal{E}_1}^{\textsf{ind-cpa-1}}(\mathcal{A})$, where $b = 1$:
$$\Pr_{\mathbf{G}_2}[E] - \Pr_{\mathbf{G}_1}[E] \leq \mathrm{Adv}_{\mathcal{E}_1}^{\textsf{ind-cpa}}(t + t_m + t_e),$$
where t_m is the time for one multiplication and t_e the time for one encryption.

Game $\mathbf{G}_3$: In this final game, the simulator plays the role of the challenger in the experiment $\mathrm{Exp}_{\mathcal{E}_T}^{\textsf{ind-cpa-1}}(\mathcal{A})$, where $b = 1$. This game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_3}[E] = \Pr_{\mathbf{G}_2}[E]$.

One can note, that in this last game, we have $\Pr_{\mathbf{G}_3}[E] = \Pr\left[\mathrm{Exp}_{\mathcal{E}_T}^{\textsf{ind-cpa-1}}(\mathcal{A}) = 1\right]$, hence
$$\begin{aligned}\Pr\left[\mathrm{Exp}_{\mathcal{E}_T}^{\textsf{ind-cpa-1}}(\mathcal{A}) = 1\right] - \Pr\left[\mathrm{Exp}_{\mathcal{E}_T}^{\textsf{ind-cpa-0}}(\mathcal{A}) = 1\right] &= \Pr_{\mathbf{G}_3}(E) - \Pr_{\mathbf{G}_0}(E)\\ &\leq \mathrm{Adv}_{\mathcal{E}_T}^{\textsf{ind-cpa}}(t + t_m + t_e),\end{aligned}$$
which concludes the proof, since it works exactly the same way for $\mathbb{G}_2$.

3.4 Homomorphic Properties

As BGN and Freeman cryptosystems, ours also supports additions, one multiplication layer, and additions. For our scheme, we detail the homomorphic functions below:

$\mathsf{Add}(C_s, C'_s)$**:** Given two ciphertexts $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s), C' = ([\mathbf{c}'_{s,1}]_s, [\mathbf{c}'_{s,2}]_s)$ in one of $\mathbb{G}_1^2 \times \mathbb{G}_1^2, \mathbb{G}_2^2 \times \mathbb{G}_2^2, \mathbb{G}_T^4 \times \mathbb{G}_T^4$, if $[\mathbf{c}_{s,2}]_s = [\mathbf{c}'_{s,2}]_s$, it outputs $([\mathbf{c}_{s,1}]_s + [\mathbf{c}'_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$, otherwise it outputs $\perp$.

$\mathsf{Multiply}(C_1, C_2)$**:** Given two ciphertexts $C_1 = ([\mathbf{c}_{1,1}]_1, [\mathbf{c}_{1,2}]_1) \in \mathbb{G}_1^2 \times \mathbb{G}_1^2$ and $C_2 = ([\mathbf{c}_{2,1}]_2, [\mathbf{c}_{2,2}]_2) \in \mathbb{G}_2^2 \times \mathbb{G}_2^2$, it outputs $C_T = ([\mathbf{c}_{1,2}]_1 \bullet [\mathbf{c}_{2,1}]_2, [\mathbf{c}_{1,2}]_1 \bullet [\mathbf{c}_{2,2}]_2) \in \mathbb{G}_T^4 \times \mathbb{G}_T^4$.

$\mathsf{Randomize}_s(\mathsf{pk}_s, C_s)$**:** Given a ciphertext $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$, for $s \in \{1, 2\}$ and a public key $\mathsf{pk}_s = [\mathbf{p}_s]_s$, it chooses $\alpha, r \xleftarrow{\$} \mathbb{Z}_p$ and outputs $(\alpha \cdot ([\mathbf{c}_{s,1}]_s + r \cdot [\mathbf{p}_s]_s), \alpha \cdot [\mathbf{c}_{s,2}]_s)$; while for $s = T$ and a public key $\mathsf{pk}_T = ([\mathbf{p}_1]_1, [\mathbf{p}_2]_2)$, it chooses $\alpha \xleftarrow{\$} \mathbb{Z}_p, [\mathbf{r}_1]_1 \xleftarrow{\$} \mathbb{G}_1^2$ and $[\mathbf{r}_2]_2 \xleftarrow{\$} \mathbb{G}_2^2$, and outputs $(\alpha \cdot ([\mathbf{c}_{T,1}]_T + [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 + [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2), \alpha \cdot [\mathbf{c}_{T,2}]_T)$.

Instead of performing a systematic randomisation of ciphertexts as proposed by Freeman each time an Add or a $\mathsf{Multiply}$ is computed, we create a specific function $\mathsf{Randomize}$ usable at any time. Thus, if it is never applied, all the operations are easily verifiable. On the contrary, if it is performed at least once just before $\mathsf{Decrypt}$, it masks all the operations. According to the privacy and verifiability requirements, one can apply it or not. Let us check the correctness of the three homomorphic functions:

Proposition 7. Add *and* $\mathsf{Multiply}$ *are correct.*

Proof. Let us first consider the addition operations:

– For $s = 1, 2$:
$$\begin{aligned}&\mathsf{Add}(\mathsf{Encrypt}_s(\mathsf{pk}_s, m, [\mathbf{a}]_s; r), \mathsf{Encrypt}_s(\mathsf{pk}_s, m', [\mathbf{a}]_s; r'))\\ &= ([m\mathbf{a} + r\mathbf{p}_s]_s \cdot [m'\mathbf{a} + r'\mathbf{p}_s]_s, [\mathbf{a}]_s) = ([(m + m')\mathbf{a} + (r + r')\mathbf{p}_s]_s, [\mathbf{a}]_s)\\ &= \mathsf{Encrypt}_s(\mathsf{pk}_s, m + m', [\mathbf{a}]_s; r + r')\end{aligned}$$

– For $s = T$:
$$\begin{aligned}&\mathsf{Add}(\mathsf{Encrypt}_T(\mathsf{pk}_T, m, ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2); \mathbf{r}_1, \mathbf{r}_2),\\ &\qquad \mathsf{Encrypt}_T(\mathsf{pk}_T, m', ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2); \mathbf{r}'_1, \mathbf{r}'_2))\\ &= ([m([\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) + \mathbf{r}_1 \otimes \mathbf{p}_2 + \mathbf{p}_1 \otimes \mathbf{r}_2]_T \cdot\\ &\qquad [m'([\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) + \mathbf{r}'_1 \otimes \mathbf{p}_2 + \mathbf{p}_1 \otimes \mathbf{r}'_2]_T, [\mathbf{a}]_1 \bullet [\mathbf{a}_2]_2)\\ &= ([(m + m')([\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) + (\mathbf{r}_1 + \mathbf{r}'_1) \otimes \mathbf{p}_2 + \mathbf{p}_1 \otimes (\mathbf{r}_2 + \mathbf{r}'_2)]_T, [\mathbf{a}]_1 \bullet [\mathbf{a}_2]_2)\\ &= \mathsf{Encrypt}_T(\mathsf{pk}_T, m + m', ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2); \mathbf{r}_1 + \mathbf{r}'_1, \mathbf{r}_2 + \mathbf{r}'_2)\end{aligned}$$

About multiplication, we can see that

$$\begin{aligned}
&\mathsf{Multiply}(\mathsf{Encrypt}_1(\mathsf{pk}_1, m_1, [\mathbf{a}_1]_1; r_1), \mathsf{Encrypt}_2(\mathsf{pk}_2, m_2, [\mathbf{a}_2]_2; r_2))\\
&= ([m_1\mathbf{a}_1 + r_1\mathbf{p}_1]_1 \cdot [m_2\mathbf{a}_2 + r_2\mathbf{p}_2]_2, [\mathbf{a}]_1 \bullet [\mathbf{a}_2]_2)\\
&= ([(m_1\mathbf{a}_1 + r_1\mathbf{p}_1) \otimes (m_2\mathbf{a}_2 + r_2\mathbf{p}_2)]_T, [\mathbf{a}]_1 \bullet [\mathbf{a}_2]_2)\\
&= ([m_1\mathbf{a}_1 \otimes m_2\mathbf{a}_2 + m_1\mathbf{a}_1 \otimes r_2\mathbf{p}_2 + r_1\mathbf{p}_1 \otimes m_2\mathbf{a}_2 + r_1\mathbf{p}_1 \otimes r_2\mathbf{p}_2]_T, [\mathbf{a}]_1 \bullet [\mathbf{a}_2]_2)\\
&= ([m_1\mathbf{a}_1 \otimes m_2\mathbf{a}_2 + m_1\mathbf{a}_1 \otimes r_2\mathbf{p}_2 + r_1\mathbf{p}_1 \otimes (m_2\mathbf{a}_2 + r_2\mathbf{p}_2)]_T, [\mathbf{a}]_1 \bullet [\mathbf{a}_2]_2)\\
&= ([m_1m_2(\mathbf{a}_1 \otimes \mathbf{a}_2) + \mathbf{p}_1 \otimes (r_1m_2\mathbf{a}_2 + r_1r_2\mathbf{p}_2) + (r_2m_1\mathbf{a}_1) \otimes \mathbf{p}_2]_T, [\mathbf{a}]_1 \bullet [\mathbf{a}_2]_2)\\
&= \mathsf{Encrypt}_T(\mathsf{pk}_T, m_1m_2, ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2); m_1r_2\mathbf{a}_1, m_2r_1\mathbf{a}_2 + r_1r_2\mathbf{p}_2)
\end{aligned}$$

Proposition 8. *For $s \in \{1, 2, T\}$,* $\mathsf{Randomize}_s$ *is correct, with $\alpha = 1$.*

Proof. For $s \in \{1, 2\}$:

$$\begin{aligned}
&\mathsf{Randomize}_s(\mathsf{pk}_s, \mathsf{Encrypt}_s(\mathsf{pk}_s, m, [\mathbf{a}]_s; r), \alpha, r')\\
&= ([\alpha(m\mathbf{a} + r\mathbf{p}_s + r'\mathbf{p}_s)]_s, [\alpha\mathbf{a}]_s) = ([m(\alpha\mathbf{a}) + \alpha(r + r')\mathbf{p}_s]_s, [\alpha\mathbf{a}]_s)\\
&= \mathsf{Encrypt}_s(\mathsf{pk}_s, m, [\alpha\mathbf{a}]_s; \alpha(r + r'))
\end{aligned}$$

Since r' is uniformly distributed, the mask of the first component of the ciphertext is uniformly distributed, as in a fresh ciphertext, while with $\alpha = 1$, the basis in the second component remains unchanged. In addition, the random α also randomizes the basis $[\alpha\mathbf{a}]_s$, in the second component of the ciphertext, but computationally only, under the DDH assumption in $\mathbb{G}_s$.

For $s = T$:

$$\begin{aligned}
&\mathsf{Randomize}_T(\mathsf{pk}_T, \mathsf{Encrypt}_T(\mathsf{pk}_T, m, ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2); r), \alpha, \mathbf{r}'_1, \mathbf{r}'_2)\\
&= (\alpha \cdot (m \cdot [\mathbf{a}]_T \boldsymbol{+} [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 \boldsymbol{+} [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2 \boldsymbol{+} [\mathbf{p}_1]_1 \bullet [\mathbf{r}'_2]_2 \boldsymbol{+} [\mathbf{r}'_1]_1 \bullet [\mathbf{p}_2]_2, [\mathbf{a}]_T),\\
&([\alpha\mathbf{a}_1]_1, [\alpha\mathbf{a}_2]_2))\\
&= (\alpha \cdot (m \cdot [\mathbf{a}]_T \boldsymbol{+} [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2 + \mathbf{r}'_2]_2 \boldsymbol{+} [\mathbf{r}_1 + \mathbf{r}'_1]_1 \bullet [\mathbf{p}_2]_2), ([\alpha\mathbf{a}_1]_1 \bullet [\alpha\mathbf{a}_2]_2))\\
&= \mathsf{Encrypt}_T(\mathsf{pk}_T, m, ([\alpha\mathbf{a}_1]_1, [\alpha\mathbf{a}_2]_2); \alpha(\mathbf{r}_2 + \mathbf{r}'_1), \alpha(\mathbf{r}_2 + \mathbf{r}'_2))
\end{aligned}$$

Again, since $\mathbf{r}'_1$ and $\mathbf{r}'_2$ are uniformly distributed, the mask of the first component of the ciphertext is uniformly distributed, as in a fresh ciphertext. In addition, the random α randomizes the basis in the second component of the ciphertext, but computationally only, under the DDH assumption in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

3.5 Re-Encryption

Now we have three efficient encryption schemes able to compute homomorphic operations and supporting multiple users. However, it is not enough to fully address our target use cases: we additionally need proxy-reencryption, in order to transform a ciphertext for Alice into a ciphertext to Bob. With the Freeman approach, and our formalism, this is just a change of basis in the exponents: we can re-encrypt a message encrypted under a private key pk^a onto another encryption for a private key pk^b by using a special secret key called re-encryption key $\mathsf{rk}^{a\to b}$. Below we describe $\mathsf{REKeygen}_s$ that creates the re-encryption key from the secret keys and $\mathsf{Re\text{-}encrypt}_s$ the function to re-encrypt a ciphertext, but under a different basis.

$\mathsf{REKeygen}_s(\mathsf{sk}^a_s, \mathsf{sk}^b_s)$: For $s = 1, 2$, from two different secret keys $\mathsf{sk}^a_s = \mathbf{P}_s$ and $\mathsf{sk}^b_s = \mathbf{P}'_s$ associated respectively to the two public keys pk^a_s and pk^b_s, compute $\mathbf{B}_s, \mathbf{B}'_s \in \mathrm{SL}_2(\mathbb{Z}_p)^2$ such that $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}\mathbf{B}_s$ and $\mathbf{P}'_s = \mathbf{B}'^{-1}_s\mathbf{U}\mathbf{B}'_s$ and output $\mathsf{rk}^{a\to b}_s = \mathbf{B}_s^{-1}\mathbf{B}'_s$ the secret re-encryption key. From the re-encryption keys $\mathsf{rk}^{a\to b}_1 \leftarrow \mathsf{REKeygen}_1(\mathsf{sk}^a_1, \mathsf{sk}^b_1)$ and $\mathsf{rk}^{a\to b}_2 \leftarrow \mathsf{REKeygen}_2(\mathsf{sk}^a_2, \mathsf{sk}^b_2)$, we will consider $\mathsf{rk}^{a\to b}_T = (\mathsf{rk}^{a\to b}_1, \mathsf{rk}^{a\to b}_2)$.

$\mathsf{Re\text{-}encrypt}_s(\mathsf{rk}_s^{a\to b}, C_s)$**:** To re-encrypt a ciphertext $C = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$:

– for $s = 1, 2$, output $([\mathbf{c}_{s,1}]_s \cdot \mathsf{rk}_s^{a\to b}, [\mathbf{c}_{s,2}]_s \cdot \mathsf{rk}_s^{a\to b})$;

– for $s = T$, output $([\mathbf{c}_{T,1}]_T \cdot (\mathsf{rk}_1^{a\to b} \otimes \mathsf{rk}_2^{a\to b}), [\mathbf{c}_{T,2}]_T \cdot (\mathsf{rk}_1^{a\to b} \otimes \mathsf{rk}_2^{a\to b}))$.

Proposition 9. $\mathsf{Re\text{-}encrypt}_s$ *is correct.*

Proof. The correctness of the re-encryption is based on a change of basis that transforms an element in the kernel of $\mathbf{P}_s$ in an element in the kernel of $\mathbf{P}'_s$: let $\mathbf{p} \in \ker(\mathbf{P}_s)$ and $\mathbf{p}' \in \ker(\mathbf{P}'_s)$ because $\ker(\mathbf{P}_s)$ and $\ker(\mathbf{P}'_s)$ are of dimension 1 in $\mathbb{Z}_p^2$, there exist $a, b, k \in \mathbb{Z}_p$, such that $\mathbf{p} = k \cdot (a, b)$ and $a', b', k' \in \mathbb{Z}_p$, such that $\mathbf{p}' = k' \cdot (a', b')$. We have:

$$\mathbf{p} \cdot \mathsf{rk} = \mathbf{p} \cdot \mathbf{B}^{-1}\mathbf{B}' = k(1, 0)\mathbf{B}' = k(a', b') \Rightarrow \mathbf{p} \cdot \mathsf{rk} = kk'^{-1}\mathbf{p}' = r'\mathbf{p}'$$

for some $r' \in \mathbb{Z}_p$ and with that, the correctness follows, where $\mathsf{rk}_s^{a\to b}$ is denoted $\mathbf{R}_s$: for $s \in \{1, 2\}$,

$$\begin{aligned}
&\mathsf{Re\text{-}encrypt}_s(\mathsf{rk}_s^{a\to b}, \mathsf{Encrypt}_s(\mathsf{pk}_s^a, m, \mathbf{a}, r)) = ([\mathbf{c}_{s,1}]_s \cdot \mathsf{rk}_s^{a\to b}, [\mathbf{c}_{s,2}]_s \cdot \mathsf{rk}_s^{a\to b})\\
&= ([m\mathbf{a}\mathbf{R}_s + r\mathbf{p}_s\mathbf{R}_s]_s, [\mathbf{a}\mathbf{R}_s]_s) = ([m\mathbf{a}\mathbf{R}_s + rr'\mathbf{p}'_s]_s, [\mathbf{a}\mathbf{R}_s]_s)\\
&= \mathsf{Encrypt}_s(\mathsf{pk}_s^b, m, \mathbf{a}\mathbf{R}_s; rr')
\end{aligned}$$

For $s = T$,

$$\begin{aligned}
&\mathsf{Re\text{-}encrypt}_T(\mathsf{rk}_T^{a\to b}, \mathsf{Encrypt}_T(\mathsf{pk}_T^a, m, \mathbf{a}; \mathbf{r}_1, \mathbf{r}_2))\\
&= ([\mathbf{c}_{T,1}]_T \cdot (\mathsf{rk}_1^{a\to b} \otimes \mathsf{rk}_2^{a\to b}), [\mathbf{c}_{T,2}]_T \cdot (\mathsf{rk}_1^{a\to b} \otimes \mathsf{rk}_2^{a\to b}))\\
&= ([(m\mathbf{a} + \mathbf{p}_1 \otimes \mathbf{r}_2 + \mathbf{r}_1 \otimes \mathbf{p}_2) \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2)]_T, [\mathbf{a} \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2)]_T)\\
&= ([m\mathbf{a}(\mathbf{R}_1 \otimes \mathbf{R}_2) + \mathbf{p}_1\mathbf{R}_1 \otimes \mathbf{r}_2\mathbf{R}_2 + \mathbf{r}_1\mathbf{R}_1 \otimes \mathbf{p}_2\mathbf{R}_2]_T, [\mathbf{a} \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2)]_T)\\
&= ([m\mathbf{a}(\mathbf{R}_1 \otimes \mathbf{R}_2) + \mathbf{r}'_1\mathbf{p}'_1 \otimes \mathbf{r}_2\mathbf{R}_2 + \mathbf{r}_1\mathbf{R}_1 \otimes \mathbf{r}'_2\mathbf{p}'_2]_T, [\mathbf{a} \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2)]_T)\\
&= ([m\mathbf{a}(\mathbf{R}_1 \otimes \mathbf{R}_2) + \mathbf{p}'_1 \otimes \mathbf{r}'_1\mathbf{r}_2\mathbf{R}_2 + \mathbf{r}'_2\mathbf{r}_1\mathbf{R}_1 \otimes \mathbf{p}'_2]_T, [\mathbf{a} \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2)]_T)\\
&= \mathsf{Encrypt}_T(\mathsf{pk}_T^b, m, \mathbf{a}(\mathbf{R}_1 \otimes \mathbf{R}_2); \mathbf{r}'_2\mathbf{r}_1\mathbf{R}_1, \mathbf{r}'_1\mathbf{r}_2\mathbf{R}_2)
\end{aligned}$$

We stress that the basis $\mathbf{a}$ is modified with the re-encryption process, into $\mathbf{a}\mathbf{R}_s$ or $\mathbf{a}(\mathbf{R}_1 \otimes \mathbf{R}_2)$, which could leak some information about the re-encryption key. But as explained above, the randomization process can provide a new ciphertext that computationally hides it, under DDH assumptions.

3.6 Verifiability

When a ciphertext is randomized or re-encrypted by a third party, one may want to be sure the content is kept unchanged. Verifiability is thus an important property we can efficiently achieve, with classical zero-knowledge proofs of discrete logarithm relations *à la* Schnorr. Such linear proofs of existence of k scalars that satisfy linear relations generally consist of a commitment c, a challenge $e \in \mathbb{Z}_p$ and the response $r \in \mathbb{Z}_p^k$ (details on the example below). The non-interactive variant just contains e and r, and thus $k + 1$ scalars.

Example 10. Let $\mathbf{M} \in \mathcal{M}_2(\mathbb{Z}_p)$ and $([\mathbf{x}]_s, [\mathbf{y}]_s), ([\mathbf{x}']_s, [\mathbf{y}']_s) \in \mathbb{G}_s^2$. We will make the zero-knowledge proof of existence of $\mathbf{M}$ such that both $[\mathbf{y}]_s = [\mathbf{x}]_s \cdot \mathbf{M}$ and $[\mathbf{y}']_s = [\mathbf{x}']_s \cdot \mathbf{M}$, where $[\mathbf{x}]_s$, $[\mathbf{y}]_s$, $[\mathbf{x}']_s$ and $[\mathbf{y}']_s$ are public, but the prover knows $\mathbf{M}$. This is the classical zero-knowledge proof of equality of discrete logarithms with matrices.

The prover chooses $\mathbf{M}' \xleftarrow{\$} \mathcal{M}_2(\mathbb{Z}_p)$ and sends the commitments $[\mathbf{c}]_s = [\mathbf{x}]_s \cdot \mathbf{M}'$ and $[\mathbf{c}']_s = [\mathbf{x}']_s \cdot \mathbf{M}'$ to the verifier that answers a challenge $e \in \mathbb{Z}_p$. The prover constructs its response $\mathbf{R} = \mathbf{M}' - e\mathbf{M}$ in $\mathcal{M}_2(\mathbb{Z}_p)$ and the verifier checks whether both $[\mathbf{c}]_s = [\mathbf{x}]_s \cdot \mathbf{R} + e[\mathbf{y}]_s$ and $[\mathbf{c}']_s = [\mathbf{x}']_s \cdot \mathbf{R} + e[\mathbf{y}']_s$, in $\mathbb{G}_s^2$. To make the proof non-interactive, one can use the Fiat-Shamir heuristic with e generated by a hash function (modeled as a random oracle) on the statement $([\mathbf{x}]_s, [\mathbf{y}]_s), ([\mathbf{x}']_s, [\mathbf{y}']_s)$ and commitments $([\mathbf{c}]_s, [\mathbf{c}']_s)$. The proof eventually consists of $(e, \mathbf{R})$. From this proof, one can compute the candidates for $([\mathbf{c}]_s, [\mathbf{c}']_s)$, and check whether the hash value gives back e.

Before entering into the details of the relations to be proven, for each function of our encryption scheme, we rewrite the Keygen_s and $\mathsf{REKeygen}_s$ algorithms to prepare the verifiability of $\mathsf{Decrypt}_s$ and $\mathsf{Re\text{-}encrypt}_s$. These new Keygen_s and $\mathsf{REKeygen}_s$ algorithms consist of the original Keygen_s and $\mathsf{REKeygen}_s$ but with more elements in the output: they both output a public version of the produced secret key plus a zero-knowledge proof of the correctness of the keys. This significantly simplifies the relations to be proven afterwards for $\mathsf{Decrypt}_s$ and $\mathsf{Re\text{-}encrypt}_s$. At the end of this section, we prove that adding those elements do not compromise the security of the encryption scheme.

Keygen_s for Verifiability. While the secret key is the projection $\mathbf{P}_s$, the verification key vsk_s consists of $[\mathbf{P}_s]_s$:

$\mathsf{Keygen}_s(\mathsf{param})$**:** For $s \in \{1,2\}$. Choose $\mathbf{B}_s \xleftarrow{\$} \mathrm{SL}_2(\mathbb{Z}_p)$. Let $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}_2\mathbf{B}_s$ and $\mathbf{p}_s \in \ker(\mathbf{P}_s) \setminus \{\mathbf{0}\}$. Output the public key $\mathsf{pk}_s = [\mathbf{p}_s]_s$, the private key $\mathsf{sk}_s = \mathbf{P}_s$ and $\mathsf{vsk}_s = [\mathbf{P}_s]_s$ a verifiable public version of the secret key with the proof π_s:

$$\{\exists \mathsf{sk}_s \in \mathcal{M}_2(\mathbb{Z}_p), \mathsf{vsk}_s \neq [\mathbf{0}]_s \wedge \mathsf{vsk}_s = [\mathbf{1}]_s \cdot \mathsf{sk}_s \wedge \mathsf{pk}_s \cdot \mathsf{sk}_s = [\mathbf{0}]_s\}.$$

The proof π_s guarantees that all the keys are well-formed: vsk_s is the exponentiation of a 2×2-matrix sk_s, for which the discrete logarithm of pk_s is in the kernel. Hence, sk_s is not full rank, and $\mathsf{vsk}_s \neq [\mathbf{0}]_s$ proves that sk_s is of dimension 1: a projection. As a consequence, π_s consists of 5 scalars of $\mathbb{Z}_p$, using the above non-interactive zero-knowledge technique *à la* Schnorr.

From $(\mathsf{vsk}_1, \mathsf{vsk}_2)$, we consider $\mathsf{vsk}_T = \mathsf{vsk}_1 \bullet \mathsf{vsk}_2$. It satisfies $\mathsf{vsk}_T = [\mathbf{P}_1 \otimes \mathbf{P}_2]_T$ if $(\mathsf{vsk}_1, \mathsf{vsk}_2) = ([\mathbf{P}_1]_1, [\mathbf{P}_2]_2)$.

$\mathsf{REKeygen}_s$ for Verifiability. As above, while the secret re-encryption key is an invertible change of basis matrix $\mathsf{rk}_s^{a\to b}$, the verification key $\mathsf{vrk}_s^{a\to b}$ consists of $[\mathsf{rk}_s^{a\to b}]_s$. But in order to prove the matrix $\mathsf{rk}_s^{a\to b}$ is invertible, one can show it is non-zero, and not of rank 1, which would mean that $\mathsf{vrk}_s^{a\to b}$ would consist of a Diffie-Hellman tuple:

$\mathsf{REKeygen}_s(\mathsf{sk}_s^a, \mathsf{sk}_s^b)$**:** For $s = 1, 2$, from two different secret keys $\mathsf{sk}_s^a = \mathbf{P}_s$ and $\mathsf{sk}_s^b = \mathbf{P}'_s$ associated respectively to the two public keys pk_s^a and pk_s^b, compute $\mathbf{B}_s, \mathbf{B}'_s \in \mathcal{M}_2(\mathbb{Z}_p)^2$ such that $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}\mathbf{B}_s$ and $\mathbf{P}'_s = {\mathbf{B}'_s}^{-1}\mathbf{U}\mathbf{B}'_s$. Let $\mathsf{rk}_s^{a\to b} = \mathbf{R}_s^{a\to b} = \mathbf{B}_s^{-1}\mathbf{B}'_s$ be the secret re-encryption key, $\mathsf{vrk}_s^{a\to b} = [\mathsf{rk}_s^{a\to b}]_s$ be a verifiable public version of the re-encryption key and $[r']_s = \lambda \cdot [r_{12}]_s$ where λ is such that $r_{21} = \lambda \cdot r_{11}$ (with $r_{11}, r_{12}, r_{21}, r_{22}$ the components of $\mathsf{rk}_s^{a\to b}$, and $\pi_s^{a\to b}$:

$$\begin{aligned}\{\exists \mathsf{rk}_s^{a\to b} \in \mathcal{M}_2(\mathbb{Z}_p), &\exists \lambda \in \mathbb{Z}_p,\\ &\mathsf{vrk}_s \neq [\mathbf{0}]_s \wedge \mathsf{vrk}_s^{a\to b} = [\mathbf{1}]_s \cdot \mathsf{rk}_s^{a\to b} \wedge \mathsf{pk}_s^b = \mathsf{pk}_s^a \cdot \mathsf{rk}_s^{a\to b}\\ &\wedge [r']_s = \lambda \cdot [r_{12}]_s \wedge [r_{21}]_s = \lambda \cdot [r_{11}]_s \wedge [r']_s \neq [r_{22}]_s\}\end{aligned}$$

Output $(\mathsf{rk}_s^{a\to b}, \mathsf{vrk}_s^{a\to b}, [r']_s, \pi_s^{a\to b})$.

The proof $\pi_s^{a\to b}$ guarantees that $\mathsf{vrk}_s^{a\to b}$ is well-formed and, since in $\mathcal{M}_2(\mathbb{Z}_p)$, the matrices are $\mathbf{0}$, or of rank 1 as a projection, or invertible: $\pi_s^{a\to b}$ first checks it is not $\mathbf{0}$, and then not of rank 1 either, as $\mathsf{vrk}_s^{a\to b}$ is not a Diffie-Hellman tuple.

Remark 11. In Keygen_s, $\mathbf{B}_s$ is taken from $\mathrm{SL}_2(\mathbb{Z}_p)$. This implies that in $\mathsf{REKeygen}_s$, $\mathsf{rk}_s^{a\to b}$ also belongs to $\mathrm{SL}_2(\mathbb{Z}_p)$. But in the verifiability, the proof, $\pi_s^{a\to b}$ just checks whether $\mathsf{rk}_s^{a\to b}$ is in $\mathrm{GL}_2(\mathbb{Z}_p)$. Actually it is enough because the only reason we take $\mathbf{B}_s \in \mathrm{SL}_2(\mathbb{Z}_p)$ is to obtain uniqueness of $\mathbf{B}_s$ from a projection $\mathbf{P}_s$. If $\mathbf{B}_s \in \mathrm{GL}_2(\mathbb{Z}_p)$ instead of $\mathrm{SL}_2(\mathbb{Z}_p)$, $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}_2\mathbf{B}_s$ is unchanged.

The two checks $\mathsf{vrk}_s^{a\to b} \neq [\mathbf{0}]_s$ and $[r']_s \neq [r_{22}]_s$ are just simple verifications, thus $\pi_s^{a\to b}$ needs 6 scalars of $\mathbb{Z}_p$ as a proof *à la* Schnorr.

Similarly as for vsk_s, from $(\mathsf{vrk}_1, \mathsf{vrk}_2)$, we consider $\mathsf{vrk}_T = \mathsf{vrk}_1 \bullet \mathsf{vrk}_2$. So that, $\mathsf{vrk}_T = [\mathbf{R}_1 \otimes \mathbf{R}_2]_T$ if $(\mathsf{vrk}_1, \mathsf{vrk}_2) = ([\mathbf{R}_1]_1, [\mathbf{R}_2]_2)$.

Now, we explain for each function, the relations to be proven:

The function Randomize$_s$. It takes a ciphertext $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$ encrypted with a public key pk_s and produces a ciphertext $C'_s = ([\mathbf{c}'_{s,1}]_s, [\mathbf{c}'_{s,2}]_s)$ such that:

– for $s \in \{1,2\}$ and $\mathsf{pk}_s = [\mathbf{p}_s]_s$, it exists $\alpha, r \in \mathbb{Z}_p$ such that:

$$[\mathbf{c}'_{s,1}]_s = \alpha \cdot ([\mathbf{c}_{s,1}]_s + r \cdot [\mathbf{p}_s]_s) \wedge [\mathbf{c}'_{s,2}]_s = \alpha \cdot [\mathbf{c}_{s,2}]_s$$

– for $s = T$ and $\mathsf{pk}_T = ([\mathbf{p}_1]_1, [\mathbf{p}_2]_2)$, it exists $\alpha \in \mathbb{Z}_p, \mathbf{r}_1, \mathbf{r}_2 \in \mathbb{Z}_p^2$ such that:

$$[\mathbf{c}'_{T,1}]_T = \alpha \cdot ([\mathbf{c}_{T,1}]_T + [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 + [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2) \wedge [\mathbf{c}'_{T,2}]_T = \alpha \cdot [\mathbf{c}_{T,2}]_T$$

These relations are equivalent to the linear relations:

– for $s \in \{1,2\}$, it exists $\alpha, r \in \mathbb{Z}_p$ such that:

$$[\mathbf{c}'_{s,1}]_s = \alpha \cdot [\mathbf{c}_{s,1}]_s + r \cdot [\mathbf{p}_s]_s \wedge [\mathbf{c}'_{s,2}]_s = \alpha \cdot [\mathbf{c}_{s,2}]_s$$

– for $s = T$, it exists $\alpha \in \mathbb{Z}_p, \mathbf{r}_1, \mathbf{r}_2 \in \mathbb{Z}_p^2$ such that:

$$[\mathbf{c}'_{T,1}]_T = \alpha \cdot [\mathbf{c}_{T,1}]_T + [\mathbf{p}_1]_T \cdot \mathbf{r}_2 + \mathbf{r}_1 \cdot [\mathbf{p}_2]_T \wedge [\mathbf{c}'_{T,2}]_T = \alpha \cdot [\mathbf{c}_{T,2}]_T$$

These proofs consist of 3 scalars of $\mathbb{Z}_p$ for $s \in \{1,2\}$, and 6 scalars of $\mathbb{Z}_p$ for $s = T$.

The functions Add and Multiply. They are public and deterministic thus everyone can check the operations.

The function Decrypt$_s$. It takes a ciphertext $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$ encrypted with a public key pk_s and produces its decryption m such that:

– for $s \in \{1,2\}$ and $\mathsf{pk}_s = [\mathbf{p}_s]_s$, it exists $\mathsf{sk}_s = \mathbf{P}_s \in \mathcal{M}_2(\mathbb{Z}_p)$ such that:

$$[\mathbf{p}_s]_s \cdot \mathbf{P}_s = [\mathbf{0}]_s \wedge \mathbf{P}_s \neq \mathbf{0} \wedge [\mathbf{c}_{s,1}]_s \cdot \mathbf{P}_s = m \cdot [\mathbf{c}_{s,2}]_s \cdot \mathbf{P}_s$$

– for $s = T$ and $\mathsf{pk}_T = ([\mathbf{p}_1]_1, [\mathbf{p}_2]_2)$, it exists $\mathsf{sk}_T = (\mathbf{P}_1, \mathbf{P}_2) \in \mathcal{M}_2(\mathbb{Z}_p)^2$ such that:

$$\begin{aligned}&[\mathbf{p}_1]_1 \cdot \mathbf{P}_1 = [\mathbf{0}]_1 \wedge [\mathbf{p}_2]_2 \cdot \mathbf{P}_2 = [\mathbf{0}]_2 \wedge \mathbf{P}_1 \neq \mathbf{0} \wedge \mathbf{P}_2 \neq \mathbf{0}\\ &\wedge [\mathbf{c}_{T,1}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) = m \cdot [\mathbf{c}_{T,2}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2)\end{aligned}$$

Instead of proving these relations, the prover will use vsk_s for $s \in \{1,2,T\}$ produced by Keygen_s for verifiability and will make the proof of the relations:

– for $s \in \{1,2\}$, it exists $\mathsf{sk}_s = \mathbf{P}_s \in \mathcal{M}_2(\mathbb{Z}_p)$ such that:

$$[\mathsf{vsk}_s]_s = [\mathbf{1}]_s \cdot \mathbf{P}_s \wedge ([\mathbf{c}_{s,1}]_s - m \cdot [\mathbf{c}_{s,2}]_s) \cdot \mathbf{P}_s = [\mathbf{0}]_s$$

– for $s = T$, it exists $\mathsf{sk}_T = (\mathbf{P}_1, \mathbf{P}_2) \in \mathcal{M}_2(\mathbb{Z}_p)^2$ such that:

$$[\mathsf{vsk}_T]_T = [\mathbf{1}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) \wedge ([\mathbf{c}_{T,1}]_T - m \cdot [\mathbf{c}_{T,2}]_T) \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) = [\mathbf{0}]_T$$

The linear proofs consist of 5 scalars of $\mathbb{Z}_p$ for $s \in \{1,2\}$ and 17 scalars of $\mathbb{Z}_p$ for $s = T$.

The function $\mathsf{Re\text{-}encrypt}_s$. It takes a ciphertext $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$ encrypted with a public key pk_s^a and produces a ciphertext $C'_s = ([\mathbf{c}'_{s,1}]_s, [\mathbf{c}'_{s,2}]_s)$ encrypted with a public key pk_s^b such that:

– for $s \in \{1,2\}$, it knows $\mathsf{rk}_s^{a\to b} = \mathbf{R}_s \in \mathrm{SL}_2(\mathbb{Z}_p)$ such that:

$$([\mathbf{c}'_{s,1}]_s, [\mathbf{c}'_{s,2}]_s) = ([\mathbf{c}_{s,1}]_s \cdot \mathbf{R}_s, [\mathbf{c}_{s,2}]_s \cdot \mathbf{R}_s) \wedge \mathsf{pk}_s^b = \mathsf{pk}_s^a \cdot \mathbf{R}_s$$

– for $s = T$, $\mathsf{pk}_T^a = (\mathsf{pk}_1^a, \mathsf{pk}_2^a)$, $\mathsf{pk}_T^b = (\mathsf{pk}_1^b, \mathsf{pk}_2^b)$ and $\mathsf{vrk}_T = ([\mathbf{R}_1]_1 \bullet [\mathbf{R}_2]_2)$, it knows $\mathsf{rk}_T^{a\to b} = (\mathbf{R}_1, \mathbf{R}_2) \in \mathrm{SL}_2(\mathbb{Z}_p)^2$ such that:

$$\begin{aligned}&([\mathbf{c}'_{T,1}]_T, [\mathbf{c}'_{T,2}]_T) = ([\mathbf{c}_{T,1}]_T \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2), [\mathbf{c}_{T,2}]_T \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2))\\ &\wedge \mathsf{pk}_T^b = (\mathsf{pk}_1^b, \mathsf{pk}_2^b) = (\mathsf{pk}_1^a \cdot \mathbf{R}_1, \mathsf{pk}_2^a \cdot \mathbf{R}_2)\end{aligned}$$

Instead of proving these relations, the prover will use vrk_s for $s \in \{1,2,T\}$ produced by $\mathsf{REKeygen}_s$ for verifiability and will make the proof of the relations below:

– for $s \in \{1,2\}$, it knows $\mathsf{rk}_s^{a\to b} = \mathbf{R}_s \in \mathcal{M}_2(\mathbb{Z}_p)$ such that:

$$([\mathbf{c}'_{s,1}]_s, [\mathbf{c}'_{s,2}]_s) = ([\mathbf{c}_{s,1}]_s \cdot \mathbf{R}_s, [\mathbf{c}_{s,2}]_s \cdot \mathbf{R}_s) \wedge \mathsf{vrk}_s^{a\to b} = [\mathbf{1}]_s \cdot \mathbf{R}_s$$

– for $s = T$, it knows $\mathsf{rk}_T^{a\to b} = (\mathbf{R}_1 \otimes \mathbf{R}_2) \in \mathcal{M}_4(\mathbb{Z}_p)$ such that:

$$\begin{aligned}&([\mathbf{c}'_{T,1}]_T, [\mathbf{c}'_{T,2}]_T) = ([\mathbf{c}_{T,1}]_T \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2), [\mathbf{c}_{T,2}]_T \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2))\\ &\wedge \mathsf{vrk}_T^{a\to b} = [\mathbf{1}]_T \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2)\end{aligned}$$

This proof needs 5 scalars of $\mathbb{Z}_p$ for $s \in \{1,2\}$ and 17 scalars of $\mathbb{Z}_p$ for $s = T$.

Proposition 12. *For $s \in \{1,2\}$, $\mathcal{E}_s$ with verifiability is still secure. More precisely, for any adversary $\mathcal{A}$ running within time t,*

$$\mathrm{Adv}_{\mathcal{E}_s}^{\mathsf{ind\text{-}cpa}}(\mathcal{A}) \leq 4 \times \mathrm{Adv}_{\mathbb{G}_s}^{\mathsf{ddh}}(t).$$

Proof. The modified Keygen_s also outputs vsk_s and a zero-knowledge proof π_s. This implies that some games need to be added before the first game in the security proof of $\mathcal{E}_s$ for Theorem 5:

Game $\mathbf{G}_0$: In the first game, the simulator plays the role of the challenger in the experiment $\mathrm{Exp}_{\mathcal{E}_s}^{\mathsf{ind\text{-}cpa\text{-}0}}(\mathcal{A})$, where $b = 0$:

$\mathcal{S}(\lambda)$:
– $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{Setup}(\lambda)$
– $(\mathsf{sk}_s, \mathsf{pk}_s, \mathsf{vsk}_s, \pi_s) \leftarrow \mathsf{Keygen}_s(\mathsf{param})$
– $m_0, m_1, [\mathbf{a}]_s \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{pk}_s)$
– $C_s = (m_0 \cdot [\mathbf{a}]_s + r \cdot [\mathbf{p}_s]_s, [\mathbf{a}]_s) \leftarrow \mathsf{Encrypt}_s(\mathsf{pk}_s, m_0, [\mathbf{a}]_s)$
– $b' \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{pk}_s, C_s)$

We are interested in the event E: $b' = 1$. By definition,

$$\Pr_{\mathbf{G}_0}[E] = \Pr\left[\mathrm{Exp}_{\mathcal{E}}^{\mathsf{ind\text{-}cpa\text{-}0}}(\mathcal{A}) = 1\right].$$

Game $\mathbf{G}_1$: The first modification is to replace π_s by its simulation, possible thanks to the zero-knowledge property. This game is statistically indistinguishable from the previous one, under the statistical zero-knowledge property of the proof *à la* Schnorr in the Random Oracle Model.

Game $\mathbf{G}_2$: Now the simulator takes as input a Diffie-Hellman tuple $([\mathbf{a}]_s, [\mathbf{b}]_s)$, with $\mathbf{b} = r \cdot \mathbf{a}$ for some scalar r, and emulates Keygen_s by defining $\mathsf{vsk}_s = ([\mathbf{a}]_s, [\mathbf{b}]_s)$. Thanks to the Diffie-Hellman tuple this corresponds to the matrix of a projection, and thus this game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_2}[E] = \Pr_{\mathbf{G}_1}[E]$.

Game $\mathbf{G}_3$: The simulator now receives a random tuple $([\mathbf{a}]_s, [\mathbf{b}]_s)$: $\Pr_{\mathbf{G}_3}[E] - \Pr_{\mathbf{G}_2}[E] \leq \mathrm{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t)$. In this game, there is no information in vsk_s anymore and the zero-knowledge proofs are simulated. In the original proof, sk_s is never used, thus we can plug the games from the original proof here. To finish the proof we need to unravel the games of vsk_s and π_s in order to have:

Game $\mathbf{G}_4$: $\mathcal{S}(\lambda)$:
- $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{Setup}(\lambda)$
- $(\mathsf{sk}_s, \mathsf{pk}_s, \mathsf{vsk}_s, \pi_s) \leftarrow \mathsf{Keygen}_s(\mathsf{param})$
- $m_0, m_1, [\mathbf{a}]_s \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{pk}_s)$
- $C_s = (m_1 \cdot [\mathbf{a}]_s + r \cdot [\mathbf{p}_s]_s, [\mathbf{a}]_s) \leftarrow \mathsf{Encrypt}_s(\mathsf{pk}_s, m_0, [\mathbf{a}]_s)$
- $b' \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{pk}_s, C_s)$

the experiment $\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_s}(\mathcal{A})$.

Hence, we have:

$$\Pr\left[\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_s}(\mathcal{A}) = 1\right] - \Pr\left[\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_s}(\mathcal{A}) = 1\right] = \Pr_{\mathbf{G}_4}(E) - \Pr_{\mathbf{G}_0}(E) \\ \leq 4 \times \mathrm{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t).$$

Corollary 13. *$\mathcal{E}_T$ with verifiability is still secure.*

Proof. Similarly to the previous proof, the zero-knowledge proofs are replaced by their simulations. Then, vsk_1 and vsk_2 are replaced by random matrices in $\mathcal{M}_2(\mathbb{Z}_p)$. Thus, vsk_T is also a random matrix.

4 Distributed Decryption

When a third-party performs the decryption, it is important to be able to prove the correct decryption, as we have shown above. But this is even better if the decryption process can be distributed among several servers, under the assumption that only a small fraction of them can be corrupted and under the control of an adversary.

In this section, we show that not only our construction handles multiple users, contrarily to BGN, but it additionally supports distributed decryption, which is definitely not the case of BGN because of the quite costly distributed generation of RSA moduli. First, we use the classical Shamir secret sharing scheme, that allows threshold decryption, but distributed generation of the keys does not look efficient. Then, we consider the particular case of 2-party decryption, and eventually present an n-party decryption, that does not need authority, even for the key generation.

4.1 Shamir Secret Sharing Scheme

To decrypt a ciphertext in $\mathbb{G}_s$ with $s \in \{1, 2\}$, one needs to compute $([\mathbf{c}_{s,1}]_s \cdot \mathsf{sk}_s, [\mathbf{c}_{s,2}]_s \cdot \mathsf{sk}_s)$. In a Shamir's like manner [Sha79], one can perform a t-out-of-n threshold secret sharing by distributing sk_s such that $\mathsf{sk}_s = \sum_{i\in I} \lambda_{I,i}\mathsf{sk}_{s,i}$ with $I \subset \{1, \dots, n\}$ a subset of t users, and for all $i \in I$, $\lambda_{I,i} \in \mathbb{Z}_p$ and $\mathsf{sk}_{s,i}$ is the secret key of the party P_i. To decrypt, all the involved parties compute $([\mathbf{x}_{1,i}]_s, [\mathbf{x}_{2,i}]_s) = ([\mathbf{c}_{s,1}]_s \cdot \mathsf{sk}_{1,i}, [\mathbf{c}_{s,2}]_s \cdot \mathsf{sk}_{1,i})$. Then, the decryption is simply the logarithm of $[\mathbf{x}_1]_s = \boldsymbol{+}_{i\in I} (\lambda_{I,i} \cdot [\mathbf{x}_{1,i}]_s)$ in base $[\mathbf{x}_2]_s = \boldsymbol{+}_{i\in I} (\lambda_{I,i} \cdot [\mathbf{x}_{2,i}]_s)$.

For $s = T$ and with just the distribution of sk_1 and sk_2, it is also possible to perform a distributed decryption. In fact, because $\mathsf{sk}_1 \otimes \mathsf{sk}_2 = (\mathsf{sk}_1 \otimes \mathbf{1}) \times (\mathbf{1} \otimes \mathsf{sk}_2)$, one can make a two round decryption by computing distributively $[\mathbf{x}_1]_T = [\mathbf{c}_{T,1}]_T \cdot (\mathsf{sk}_1 \otimes \mathbf{1})$ then $[\mathbf{y}_1]_T = [\mathbf{x}_1]_T \cdot (\mathbf{1} \otimes \mathsf{sk}_2)$.

Remark 14. Because the operations to decrypt or re-encrypt are the same, one can make distributed re-encryption in the same vein.

While verifiability could work as for simple decryption, the secret key must be a projection matrix, which is not easy to generate at random: a central authority is required here.

4.2 Two-Party Decryption

Our second approach follows the remark: $\mathsf{sk}_1 \otimes \mathsf{sk}_2 = (\mathsf{sk}_1 \otimes \mathbf{1}) \times (\mathbf{1} \otimes \mathsf{sk}_2)$. We assume the special case where a user U_1 knows sk_1 and a user U_2 knows sk_2. From local or external input ciphertext, they can come up with a final ciphertext $C_T = ([\mathbf{c}_{T,1}]_T, [\mathbf{c}_{T,2}]_T)$. Before operating the decryption, since they may have introduce their own random coins in some input ciphertexts, the final ciphertext must be refreshed to hide any leakage about combinations between messages and random coins. Each party thus applies $\mathsf{Randomize}_T$. Then, they can iteratively decrypt: party U_1 begins the decryption by computing $C'_T = ([\mathbf{c}'_{T,1}]_T, [\mathbf{c}'_{T,2}]_T) = ([\mathbf{c}_{T,1}]_T \cdot (\mathsf{sk}_1 \otimes \mathbf{1}), [\mathbf{c}_{T,2}]_T \cdot (\mathsf{sk}_1 \otimes \mathbf{1}))$ and U_2 can finish the decryption with its secret key $(\mathbf{1} \otimes \mathsf{sk}_2)$. U_2 is the first to get the result.

The generation of the keys sk_1 and sk_2 can be performed independently, and so no authority is needed, but we are limited to two-party decryption only.

To highlight the importance of the application of the function $\mathsf{Randomize}$ before starting the decryption, we detail a specific attack on the example 15. The idea is that the first decrypting party corresponds to a decrypting oracle for the other and thus can leak some information if the random part is not refreshed.

Example 15. Consider the situation where a user U_1 encrypts a message $m_1 \in \{0,1\}$ with $[\mathbf{a}_1]_1$ and r_1, and U_2 encrypts a message $m_2 \in \{0,1\}$ with $[\mathbf{a}_2]_2$ and r_2. We denote by $C_1 = ([\mathbf{c}_{1,1}]_1, [\mathbf{c}_{1,2}]_1)$ the first ciphertext and by $C_2 = ([\mathbf{c}_{2,1}]_2, [\mathbf{c}_{2,2}]_2)$ the second. Suppose U_1 and U_2 want to compute the multiplication of their inputs (which corresponds to an AND):

$$\begin{aligned} C_T = ([\mathbf{c}_{T,1}]_T, [\mathbf{c}_{T,2}]_T) &= \mathsf{Multiply}(C_1, C_2) \\ &= (m_1 m_2 \cdot [\mathbf{a}_1 \otimes \mathbf{a}_2]_T + [\mathbf{r}]_1 \bullet [\mathbf{p}_2]_2 + [\mathbf{p}_1]_1 \bullet [\mathbf{r}']_2, [\mathbf{a}_1 \otimes \mathbf{a}_2]_T) \end{aligned}$$

with $\mathbf{r} = m_1 r_2 \mathbf{a}_1$ and $\mathbf{r}' = m_2 r_1 \mathbf{a}_2 + r_1 r_2 \mathbf{p}_2$. In the case where $m_1 = 0$, U_1 may want that U_2 begins the decryption to learn m_2, even if he knows the final result will be 0. Indeed, if $m_1 = 0$, $C_T = ([\mathbf{p}_1]_1 \bullet [m_2 r_1 \mathbf{a}_2 + r_1 r_2 \mathbf{p}_2]_2, [\mathbf{0}]_T)$ and the partial decryption $C'_T = ([\mathbf{p}_1]_1 \bullet [m_2 r_1 \mathbf{a}_2]_2 \cdot (I_2 \otimes \mathbf{P}_2), [\mathbf{0}]_T)$. The problem is that $C'_T = ([\mathbf{0}]_T, [\mathbf{0}]_T)$ if $m_2 = 0$ and is different to 0 otherwise: U_1 learns the input of U_2.

To address this issue, this is enough to randomize C_T before decryption, because then the ciphertext becomes, in the case $m_1 = 0$,

$$(\alpha \cdot ([\mathbf{p}_1]_1 \bullet [m_2 r_1 \mathbf{a}_2 + r_1 r_2 \mathbf{p}_2]_2) + [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2 + [\mathbf{p}_1]_1 \bullet [\mathbf{r}'_2]_2, [\mathbf{0}]_T)$$

which is decrypted as $C'_T = ([\mathbf{p}_1]_1 \bullet [\alpha m_2 r_1 \mathbf{a}_2 + \mathbf{r}'_2]_2 \cdot (I_2 \otimes \mathbf{P}_2), [\mathbf{0}]_T)$ by U_2: it does not depend anymore on the value of m_2. Actually, instead of performing a complete $\mathsf{Randomize}$, U_2 can just randomize his random part (the part in $\mathbb{G}_2$). Hence, the random in C_T will be refreshed. One can note that, in case of an Add, the randoms are added and thus this concern does not arise.

4.3 Multi-Party Decryption

The third technique will exploit the proxy re-encryption process to offer an n-party decryption. The idea is to create a ring along which each participant re-encrypts a ciphertext until they all can decrypt the final ciphertext.

$\mathsf{Keygen}_s(\mathsf{param}, \mathcal{P}_1, \ldots, \mathcal{P}_n)$**:** For $s \in \{1,2\}$, we let $\mathsf{sk}_{0,s} = \mathbf{P}_{0,s} = \mathbf{U}_2$ and $\mathsf{pk}_{0,s} = ([1]_s, [0]_s)$ the initial keys. Then, along a ring, for $i = 1, \ldots, n$, $\mathcal{P}_i$ chooses $\mathsf{rk}_{i,s} \xleftarrow{\$} \mathrm{SL}_2(\mathbb{Z}_p)$ its secret re-encryption key and defines $\mathbf{P}_{i,s} = \mathsf{rk}_{i,s}^{-1} \mathbf{P}_{i-1,s} \mathsf{rk}_{i,s}$ and $\mathsf{pk}_{i,s} = \alpha_{i,s} \cdot \mathsf{pk}_{i-1,s} \cdot \mathsf{rk}_{i,s}$, with $\alpha_i \xleftarrow{\$} \mathbb{Z}_p^*$. The public key is then $\mathsf{pk}_s = \mathsf{pk}_{n,s}$. For $s = T$, $\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2)$.

$\mathsf{Decrypt}_s(\mathsf{rk}_{1,s}, \ldots, \mathsf{rk}_{n,s}, C_s)$**:** Since the ciphertext is encrypted under the last key on the ring pk_s, for $s = 1, 2$, $C_s = ([\mathbf{c}_{n,s,1}]_s, [\mathbf{c}_{n,s,2}]_s)$, and so each user around the ring, for $i = n, \ldots, 1$, can re-encrypt for the previous user: $([\mathbf{c}_{i-1,s,1}]_s, [\mathbf{c}_{i-1,s,2}]_s) = ([\mathbf{c}_{i,s,1}]_s \cdot \mathsf{rk}_{i,s}^{-1}, [\mathbf{c}_{i,s,2}]_s \cdot \mathsf{rk}_{i,s}^{-1})$, until $C'_s = ([\mathbf{c}_{0,s,1}]_s, [\mathbf{c}_{0,s,2}]_s)$, a ciphertext encrypted under $\mathsf{pk}_{0,s} = ([1]_s, [0]_s)$, that anybody can decrypt using $\mathsf{sk}_{0,s} = \mathbf{P}_{0,s} = \mathbf{U}_2$. For $s = T$, one does the same with $\mathsf{rk}_{i,1} \otimes \mathsf{rk}_{i,2}$.

Remark 16. For clarity, we can explicit the change of basis matrix $\mathbf{B}_{i,s}$ associated to $\mathbf{P}_{i,s}$: $\mathbf{B}_{0,s} = \mathbf{I}_2$ and $\mathbf{B}_{i,s} = \mathbf{B}_{i-1,s}\mathsf{rk}_{i,s}$.

Remark 17. The verifiability of the distributed decryption is obtained by applying successive re-encryption verifiability.

Proposition 18. *The public keys generated with the distributed* Keygen_s *protocol have the same distribution as the public keys generated in the original encryption scheme.*

Proof. Recall that Keygen_s defines keys for $s \in \{1, 2\}$. In the original encryption scheme, the secret key $\mathsf{sk}_s = \mathbf{P}_s$ is defined with a randomly chosen matrix $\mathbf{B}_s \overset{\$}{\leftarrow} \mathrm{SL}_2(\mathbb{Z}_p)$ as $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}_2\mathbf{B}_s$ and the public key $\mathsf{pk}_s = [\mathbf{p}_s]_s$ is a vector in the kernel of $\mathbf{P}_s$. Because of the equality $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}_2\mathbf{B}_s$, $\mathbf{p}_s \in \ker(\mathbf{P}_s) \Leftrightarrow \exists\alpha \in \mathbb{Z}_p^*, \mathbf{p}_s = \alpha \cdot (1, 0) \times \mathbf{B}_s$. For our n-party public key $\mathsf{pk}_s = \mathsf{pk}_{n,s} = (\alpha_{n,s} \times \ldots \times \alpha_{1,s}) \cdot ([1]_s, [0]_s) \cdot (\mathsf{rk}_{1,s} \times \ldots \times \mathsf{rk}_{n,s})$. Thus pk_s belongs to the same distribution as the public keys generated by Keygen_s with $\mathbf{B}_s = \mathsf{rk}_{1,s} \times \ldots \times \mathsf{rk}_{n,s}$ a random matrix in $\mathrm{SL}_2(\mathbb{Z}_p)$.

Proposition 19. *The distributed* $\mathsf{Decrypt}_s$ *protocol is correct.*

Proof. Each step of the decryption is actually a re-encryption under $\mathsf{pk}_{i-1,s}$, until $\mathsf{pk}_{0,s}$. Hence the correctness.

Proposition 20. *The distributed* $\mathsf{Decrypt}_s$ *protocol is secure.*

Proof. The goal of this proof is to show that the distributed protocol does not leak more information than the direct decryption would do: the plaintext only. To this aim, we assume a few players to be corrupted, and so the simulator knows the public key pk_s, the corrupted keys $\mathsf{rk}_{i,s}$, for $i \in \mathcal{C}$ (the set of the corrupted players), and the pair $(m, C_s = ([\mathbf{c}_{n,s,1}]_s, [\mathbf{c}_{n,s,2}]_s))$ of a plaintext-ciphertext. From this information, it can simulate all the honest players, without their secret keys: first, one generates a random ciphertext $([\mathbf{c}_{0,s,1}]_s, [\mathbf{c}_{0,s,2}]_s))$ of m under $\mathsf{pk}_{0,s} = ([1]_s, [0]_s)$; if $n \in \mathcal{C}$, one can compute $([\mathbf{c}_{n-1,s,1}]_s, [\mathbf{c}_{n-1,s,2}]_s)) = [\mathbf{c}_{n,s,1}]_s \cdot \mathsf{rk}_{n,s}^{-1}, [\mathbf{c}_{n,s,2}]_s \cdot \mathsf{rk}_{n,s}^{-1})$, and so on, until $([\mathbf{c}_{j,s,1}]_s, [\mathbf{c}_{j,s,2}]_s)$ for $j \notin \mathcal{C}$; if $1 \in \mathcal{C}$, one can compute $([\mathbf{c}_{1,s,1}]_s, [\mathbf{c}_{1,s,2}]_s) = ([\mathbf{c}_{0,s,1}]_s \cdot \mathsf{rk}_{1,s}, [\mathbf{c}_{0,s,2}]_s \cdot \mathsf{rk}_{1,s})$; and so one, until $([\mathbf{c}_{i,s,1}]_s, [\mathbf{c}_{i,s,2}]_s)$ for $j \notin \mathcal{C}$; until $i + 1 \notin \mathcal{C}$.

If $i + 1 = j$, this is the unique non-corrupted player, and so all the intermediate ciphertexts have been generated, correctly. If $i + 1 < j$, then $([\mathbf{c}_{i+1,s,1}]_s, [\mathbf{c}_{i+1,s,2}]_s)$ is chosen at random, etc according whether $\mathsf{rk}_{k,s}$ is known or not.

Essentially, when the re-encryption key is known (below or above), it is used, otherwise the ciphertext is chosen at random. Under the DDH assumption, these random ciphertexts are indistinguishable from the correct values. Even in case of verifiability, fake zero-knowledge proofs can be simulated for the honest users.

5 Applications

5.1 Encryption for Quadratic Multivariate Polynomials

In some cases of application, the inputs need to belong to a small space I. Boneh et al. suggested in [BGN05] a way to guarantee the ciphertext is an encryption of an input in I: if x needs to belong to I, it suffices to check if x is a root of the polynomial $\mathit{\Pi}_I(x) = \prod_{i \in I}(x - i)$. In the case where I is a 2-tuple as $\{0, 1\}$, the evaluation on $\mathit{\Pi}_I$ can be done with our scheme (or the one of BGN or Freeman).

The example given in [BGN05] is the case of electronic voting where the result to compute is simply the sum of the inputs. Since the vote needs to be an encryptions of either a 0 or a 1, the solution is to introduce the additional term $\mathsf{Add}_j(\mathsf{Multiply}(C_{x_j}, \mathsf{Add}(C_{x_j}, C_{-1}))$. For this case of application, we add a quadratic term to a linear function, just for verification. Below, we propose two cases of applications that are already quadratic, thus the verification does not add any extra cost. Note that our scheme is more suitable for electronic voting since it offers decentralization which is a fundamental notion for this example.

5.2 Encryption for Boolean Formulae

In this part, we detail the specific case of the evaluation of 2-DNF by first recalling the definition of 2-DNF, explaining how to consider them to apply our scheme and finally by rewriting the entire scheme with simplifications for this specific case. Especially, the ciphertexts will not be pairs anymore, as the second term can be removed, and we will explain why.

Every Boolean formula can be expressed as a disjunction of conjunctive clauses (an OR of ANDs). This form is called disjunctive normal form (DNF) and, more precisely, k-DNF when each clause contains at most k literals. Thus, a 2-DNF formula over the variables $x_1, \ldots, x_n \in \{0,1\}$ is of the form

$$\bigvee_{i=1}^{m} (\ell_{i,1} \wedge \ell_{i,2}) \text{ with } \ell_{i,1}, \ell_{i,2} \in \{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}.$$

The conversion of 2-DNF formulae into multivariate polynomials of total degree 2 is simple: given $\Phi(x_1, \ldots, x_n) = \bigvee_{i=1}^{m} (\ell_{i,1} \wedge \ell_{i,2})$ a 2-DNF formula, define $\phi(x_1, \ldots, x_n) = \sum_{i=1}^{m} (y_{i,1} \times y_{i,2})$ where $y_{i,j} = \ell_{i,j}$ if $\ell_{i,j} \in \{x_1, \ldots, x_n\}$ or $y_{i,j} = (1 - \ell_{i,j})$ otherwise. In this conversion, a true literal is replaced by 1, and a false literal by 0. Then, an OR is converted into an addition, and an AND is converted into a multiplication. A NOT is just $(1-x)$ when $x \in \{0,1\}$. $\phi(x_1, \ldots, x_n)$ is the multivariate polynomial of degree 2 corresponding to $\Phi(x_1, \ldots, x_n)$. As just said, this conversion works if for the inputs, we consider $1 \in \mathbb{Z}_p$ as true and $0 \in \mathbb{Z}_p$ as false, but for the output, $0 \in \mathbb{Z}_p$ is still considered as false whereas any other non-zero value is considered as true.

To evaluate the 2-DNF in an encrypted manner, we propose to encrypt the data and to calculate the quadratic polynomial corresponding to the 2-DNF as seen above by performing Adds and $\mathsf{Multiply}$s. Because the result of the 2-DNF is a Boolean, when a decryption is performed, if the result is equal to 0, one can consider it corresponds to the 0-bit (false) and else, it corresponds to the 1-bit (true).

Hence, when encrypting bits, we propose two different encodings before encryption, depending on the situation: either the 0-bit (false) is encoded by $0 \in \mathbb{Z}_p$ and the 1-bit (true) is encoded by any non-zero integer of $\mathbb{Z}_p^*$; or the 0-bit (false) is encoded by $0 \in \mathbb{Z}_p$ and the 1-bit (true) is encoded by $1 \in \mathbb{Z}_p$. With this second solution, it offers the possibility to perform one NOT on the data before Adds and $\mathsf{Multiply}$s by the operation $1 - x$. However, one has to be aware of making $\mathsf{Randomize}$ before decryption to mask the operations but also the input data in some situations: for example, if an Add is performed between three 1s, the result 3 leaks information and needs to be randomized.

Now, we will detail possible simplifications of the scheme. First, because one just wants to know whether the result is equal to 0 or the result is different from 0, we do not need $[\mathbf{a}_s]_s$ anymore: we can decrypt by just checking whether $[\mathbf{c}_s]_s \cdot \mathsf{sk}_s = [\mathbf{0}]_s$ or not. This implies that the second term of the ciphertext can be omitted. Moreover, we can fix $[\mathbf{a}_s]_s = [(1,0)]_s$ for $s \in \{1,2\}$ and $[\mathbf{a}_T]_T = [(1,0,0,0)]_s$ by taking care that the first line of $\mathbf{B}_s$ does not have 0-component in the first line in order to be sure that $\mathbf{a}_s = (1,0)$ is not in the kernel of $\mathbf{P}_s$. The rewriting scheme with simplifications is:

$\mathsf{Setup}(\lambda)$**:** Given a security parameter $\lambda \in \mathbb{N}$, run and output

$$\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathcal{G}(\lambda).$$

$\mathsf{Keygen}_s(\mathsf{param})$**:** For $s \in \{1,2\}$. Choose $\mathbf{B}_s \overset{\$}{\leftarrow} \mathrm{SL}_2(\mathbb{Z}_p)$ without any 0-component in the first line. Let $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}_2\mathbf{B}_s$ and $\mathbf{p}_s \in \ker(\mathbf{P}_s) \setminus \{\mathbf{0}\}$. Output the public key $\mathsf{pk}_s = [\mathbf{p}_s]_s$ and the private key $\mathsf{sk}_s = \mathbf{P}_s$.

From $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Keygen}_1(\mathsf{param})$ and $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{Keygen}_2(\mathsf{param})$, one can consider $\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathsf{sk}_T = (\mathsf{sk}_1, \mathsf{sk}_2)$, which are associated public and private keys in $\mathbb{G}_T$.

$\mathsf{Encrypt}_s(\mathsf{pk}_s, m)$**:** For $s \in \{1,2\}$, to encrypt a message $m \in \{0,1\}$ using public key $\mathsf{pk}_s = [\mathbf{p}_s]_s$, choose $r \xleftarrow{\$} \mathbb{Z}_p$ and output the ciphertext $C_s = m \cdot [(1,0)]_s + r \cdot [\mathbf{p}_s]_s \in \mathbb{G}_s^2$.

For $s = T$, choose $[\mathbf{r}_1]_1 \xleftarrow{\$} \mathbb{G}_1^2, [\mathbf{r}_2]_2 \xleftarrow{\$} \mathbb{G}_2^2$, and output

$$C_T = m \cdot [(1,0,0,0)]_T + [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 + [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2 \in \mathbb{G}_T^4.$$

$\mathsf{Decrypt}_s(\mathsf{sk}_s, C_s)$**:** For $s \in \{1,2,T\}$, given $C_s = [\mathbf{c}_s]_s$ and $\mathsf{sk}_s = \mathbf{P}_s$ (if $s = T$ let $\mathbf{P}_T = \mathbf{P}_1 \otimes \mathbf{P}_2$), let $C'_s = [\mathbf{c}_s]_s \cdot \mathbf{P}_s$ and output 0 if $C'_s = [\mathbf{0}]_s$ and 1 otherwise.

5.3 Group Testing on Encrypted Data

In this application we assume that a hospital collects some blood samples and wants to check which samples are positive or negative to a specific test. Group testing [Dor43] is an efficient technique to detect positive samples with fewer tests in the case the proportion of positive cases is small. The technique consists in mixing some samples, and to perform tests on fewer mixes. More precisely, we denote $\mathbf{X} = (x_{ij})$ the matrix of the mixes: $x_{ij} = 1$ if the i-th sample is in the j-th mix, otherwise $x_{ij} = 0$. The hospital then sends the (blood) mixes to a laboratory for testing them: we denote y_j the result of the test on the j-th mix.

If a patient (its sample) is in a mix with a negative result, he is negative (not infected). If a patient (its sample) is in a mix with a positive result, we cannot say anything. However, for well-chosen parameters, if a patient is not declared negative, he is likely positive. Thus, for a patient i, the formula that we want to solve is $\neg\mathsf{F}_i(\mathbf{X}, \mathbf{y}) = \bigvee_j (x_{ij} \wedge \neg y_j)$ which means that it exists a mix containing the i-th sample for which the test is negative, the global formula is negative (false). The matrix $\mathbf{X}$ of the samples needs to be encrypted since the patient does not want the laboratory to know his result. Because of the sensitiveness of the data, the result of the tests needs to be encrypted too. But the patient will need access to his own result.

In this scenario, the hospital computes for all i, j, $C_{x_{ij}} \in \mathbb{G}_1^2$, the encryption of x_{ij}, and the laboratory computes for all j, $C_{\overline{y_j}}$, the encryption of $\overline{y_j}$ in $\mathbb{G}_2^2$. Then, they both send the ciphertexts to an external database. With our encryption scheme, to compute $\neg\mathsf{F}_i$, we need to use the homomorphic properties:

$$C_i = \mathsf{Randomize}(\mathsf{Add}_j(\mathsf{Multiply}(C_{x_{ij}}, C_{\overline{y_j}}))$$

An external controller can verify the computations and if it is correct, it performs a 2-party decryption from Section 4.3 with a patient to get $\neg\mathsf{F}_i$, and thus F_i. In this way, the patient cannot decrypt the database or the result of the tests directly, but only with the help of a (possibly distributed) controller. The goal of this controller is to limit access to the specific users only. Under an assumption about the collusions, nobody excepted the users will have access to the results.

5.4 Machine Learning on Encrypted Data

Another famous applications is machine learning, where we have some trainers that fill a database and users who want to know a function of their inputs and the database. For privacy reasons, trainers do not want the users to learn the training set, and users do not want the trainers to learn their inputs. As in the previous case, we will involve a controller, as a (possibly distributed) third party to limit decryptions, but the controller should not learn anything either.

Suppose in a very large network of nodes in which some combinations should be avoided as they would result to failures. When a failure happens, the combination is stored in a database. And before applying a given combination, one can check it will likely lead to a failure, and then change. For example, the network can be a group of people where each of them can receive data. But, for some specific reasons, if a subgroup A of people is knowing a file a, the subgroup B

must not have the knowledge of a file b. This case of application can be view as a consistency model [Sch14] which can be formally described as: the input is a vector of states (each being either true or false), and if in the database all the j-th states are true a new input needs to have its j-th state to be true; if all the j-th states in the database are false, the new input needs to have its j-th state to be false; otherwise the j-th state can be either true or false. As a consequence, if we denote the i-th element of the database as a vector $\mathbf{x}_i = (x_{ij})_j$ and the user's vector by $\mathbf{y} = (y_j)$, that vector $\mathbf{y}$ is said consistent with the database the following predicate is true:

$$\bigwedge_j \Big((\wedge_i x_{ij} \wedge y_j) \vee (\wedge_i \overline{x_{ij}} \wedge \overline{y}_j) \vee (\vee_i x_{ij} \wedge \vee_i \overline{x_{ij}}) \Big) .$$

Let $X_j = \wedge_i x_{ij}$, $Y_j = \wedge_i \overline{x_{ij}}$, and $Z_j = \vee_i x_{ij} \wedge \vee_i \overline{x_{ij}}$. We define $\mathsf{F}(\mathbf{x}_1, \ldots, \mathbf{x}_m, \mathbf{y})$ the formula that we want to compute on the encrypted:

$$\mathsf{F}(\mathbf{x}_1, \ldots, \mathbf{x}_m, \mathbf{y}) = \bigwedge_j \Big((X_j \wedge y_j) \vee (Y_j \wedge \overline{y}_j) \vee Z_j \Big).$$

By definition, X_j, Y_j, and Z_j are exclusive, as X_j means the literals are all true, Y_j means the literals are all false, and Z_j means there are both true and false literals. So we have: $X_j \vee Z_j = \overline{Y_j}$ and $Y_j \vee Z_j = \overline{X_j}$. Thus, we have

$$\neg\mathsf{F}(\mathbf{x}_1, \ldots, \mathbf{x}_m, \mathbf{y}) = \bigvee_j \left((Y_j \vee \overline{y_j}) \wedge (X_j \vee y_j) \right).$$

Now, we will see how the encryption and the decryption is performed to obtain the result of an evaluation.

First, we explain how the trainers can update the database, when adding a vector $\mathbf{x}_m$. The values X_j are updated into X'_j as

$$X'_j = \bigwedge_{i=1}^{m} x_{ij} = \bigwedge_{i=1}^{m-1} x_{ij} \wedge x_{mj} = \begin{cases} X_j = \wedge_{i=1}^{m-1} x_{ij} & \text{if } x_{mj} = \mathsf{true} \\ \mathsf{false} & \text{otherwise} \end{cases}$$

which is easy to compute for the trainer, since it knows $\mathbf{x}_m$ in clear, even if X_j is encrypted: the trainer can dynamically compute C_{X_j} the encryption of X_j, when adding a new line in the database, by just making a $\mathsf{Randomize}$ if x_{mj} is true (to keep the value X_j unchanged), or by replacing the value by a fresh encryption of 0 otherwise. Similarly, the trainer can update C_{Y_j}, the encryption of Y_j.

On the user-side, he can compute C_{y_j} and $C_{\overline{y_j}}$ the encryptions of his inputs y_j and $\overline{y_j}$ respectively. Then, everyone and thus the controller itself can compute:

$$C_j = \mathsf{Randomize}\Big(\mathsf{Add}_j\Big(\mathsf{Multiply}(\mathsf{Add}(C_{Y_j}, C_{\overline{y_j}}), \mathsf{Add}(C_{X_j}, C_{y_j}))\Big)\Big).$$

Because of the $\mathsf{Multiply}$, C_{Y_j} and $C_{\overline{y_j}}$ must be ciphertexts in $\mathbb{G}_1$, while C_{X_j} and C_{y_j} must be ciphertexts in $\mathbb{G}_2$. To allow a control of the final decryption, a distributed controller can help the user to decrypt, along the scenario described in Section 4.3.

6 Optimization

In this section we propose a slight improvement by using specific orthogonal projections. This way, the projection matrix (the secret key) is defined by just one vector $\mathbf{s}$ that spans the image of the projection, and the public key $[\mathbf{p}]$ is derived from an orthogonal vector $\mathbf{p}$ that will cancel the former vector with an inner product: $\mathbf{p}$ spans the kernel of the projection. Before presenting the improvement, we recall some properties of the inner product between two vectors, that is defined as $\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a} \cdot \mathbf{b}^T$:

Proposition 21. *For any* $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \in \mathbb{Z}_p^2, \mathbf{M} \in \mathcal{M}_2(\mathbb{Z}_p)$,

$$\langle \mathbf{a} \otimes \mathbf{b}, \mathbf{c} \otimes \mathbf{d} \rangle = \langle \mathbf{a}, \mathbf{c} \rangle \otimes \langle \mathbf{b}, \mathbf{d} \rangle \tag{1}$$

$$\langle \mathbf{a}, \mathbf{b}\mathbf{M} \rangle = \langle \mathbf{a}\mathbf{M}^T, \mathbf{b} \rangle \tag{2}$$

Proof. Since $(\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T$, we have:

$$\begin{aligned}\langle \mathbf{a} \otimes \mathbf{b}, \mathbf{c} \otimes \mathbf{d} \rangle &= (\mathbf{a} \otimes \mathbf{b}) \cdot (\mathbf{c} \otimes \mathbf{d})^T = (\mathbf{a} \otimes \mathbf{b}) \cdot (\mathbf{c}^T \otimes \mathbf{d}^T) \\ &= (\mathbf{a} \cdot \mathbf{c}^T) \otimes (\mathbf{b} \cdot \mathbf{d}^T) = \langle \mathbf{a}, \mathbf{c} \rangle \otimes \langle \mathbf{b}, \mathbf{d} \rangle\end{aligned}$$

which proves the first relation. And the second relation comes from the definition of the inner product: $\langle \mathbf{a}, \mathbf{b}\mathbf{M} \rangle = \mathbf{a} \cdot (\mathbf{b}\mathbf{M})^T = \mathbf{a} \cdot \mathbf{M}^T \cdot \mathbf{b}^T = \langle \mathbf{a}\mathbf{M}^T, \mathbf{b} \rangle$.

Two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^2$ are said orthogonal if $\langle \mathbf{a}, \mathbf{b} \rangle = 0$. Then $\langle [\mathbf{a}]_s, \mathbf{b} \rangle = [\mathbf{0}]_s$ for $s \in \{1, 2\}$. Obviously, we have the same relation for $s = T$. Let us now describe our optimization. Only the secret key is different, and so the Keygen_s and $\mathsf{Decrypt}_s$ algorithms:

$\mathsf{Setup}(\lambda)$**:** Given a security parameter $\lambda \in \mathbb{N}$, run and output

$$\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathcal{G}(\lambda).$$

$\mathsf{Keygen}_s(\mathsf{param})$**:** For $s \in \{1, 2\}$. Choose $\mathbf{p}_s \overset{\$}{\leftarrow} \mathbb{Z}_p^2$ and $\mathbf{s}_s \in \mathbb{Z}_p^2$ such that $\langle \mathbf{p}_s, \mathbf{s}_s \rangle = \mathbf{0}$ and output the public key $\mathsf{pk}_s = [\mathbf{p}_s]_s$ and the private key $\mathsf{sk}_s = \mathbf{s}_s$. From $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Keygen}_1(\mathsf{param})$ and $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{Keygen}_2(\mathsf{param})$, one can consider $\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathsf{sk}_T = (\mathsf{sk}_1, \mathsf{sk}_2)$, which are associated public and private keys in $\mathbb{G}_T$.

$\mathsf{Encrypt}_s(\mathsf{pk}_s, m, A_s)$**:** (exactly as in the original protocol) For $s \in \{1, 2\}$, to encrypt a message m using public key pk_s and $A_s = [\mathbf{a}]_s \in \mathbb{G}_s^2$, choose $r \overset{\$}{\leftarrow} \mathbb{Z}_p$ and output the ciphertext $C_s = (m \cdot [\mathbf{a}]_s + r \cdot [\mathbf{p}_s]_s, [\mathbf{a}]_s) \in \mathbb{G}_s^2 \times \mathbb{G}_s^2$.

For $s = T$, with $A_s = ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2)$, set $[\mathbf{a}]_T = [\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2 \in \mathbb{G}_T^4$, choose $[\mathbf{r}_1]_1 \overset{\$}{\leftarrow} \mathbb{G}_1^2, [\mathbf{r}_2]_2 \overset{\$}{\leftarrow} \mathbb{G}_2^2$, and output $C_T = (m \cdot [\mathbf{a}]_T + [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 + [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2, [\mathbf{a}]_T) \in \mathbb{G}_T^4 \times \mathbb{G}_T^4$.

$\mathsf{Decrypt}_s(\mathsf{sk}_s, C_s)$**:** For $s \in \{1, 2\}$, given $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$ and $\mathsf{sk}_s = \mathbf{s}_s$, let $C'_s = (\langle [\mathbf{c}_{s,1}]_s, \mathbf{s}_s \rangle, \langle [\mathbf{c}_{s,2}]_s, \mathbf{s}_s \rangle)$. For $s = T$, compute $C'_T = (\langle [\mathbf{c}_{T,1}]_T, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle, \langle [\mathbf{c}_{T,2}]_T, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle)$. In both cases, output the logarithm of the first component of $\mathbf{c}'_{s,1}$ in basis the second component of $\mathbf{c}'_{s,2}$.

Proposition 22. *For* $s \in \{1, 2, T\}$*, this scheme is correct.*

Proof. For $s \in \{1, 2\}$:

$$\begin{aligned}\langle [\mathbf{c}_{s,2}]_s, \mathbf{s}_s \rangle &= \langle [\mathbf{a}_s]_s, \mathbf{s}_s \rangle \\ \langle [\mathbf{c}_{s,1}]_s, \mathbf{s}_s \rangle &= \langle m \cdot [\mathbf{a}]_s + r \cdot [\mathbf{p}_s]_s, \mathbf{s}_s \rangle = \langle m \cdot [\mathbf{a}]_s, \mathbf{s}_s \rangle + \langle r \cdot [\mathbf{p}_s]_s, \mathbf{s}_s \rangle \\ &= \langle m \cdot [\mathbf{a}]_s, \mathbf{s}_s \rangle + [\mathbf{0}]_s = m \cdot \langle [\mathbf{a}]_s, \mathbf{s}_s \rangle = m \cdot \langle [\mathbf{c}_{s,2}]_s, \mathbf{s}_s \rangle\end{aligned}$$

For $s = T$, using equation 1:

$$\begin{aligned}&\langle [\mathbf{c}_{T,2}]_T, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle = \langle [\mathbf{a}_T]_T, , \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle \\ &\langle [\mathbf{c}_{T,1}]_T, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle = \langle m \cdot [\mathbf{a}]_T + [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 + [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle \\ &= \langle m \cdot [\mathbf{a}]_T, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle + \langle [\mathbf{r}_1]_1 \otimes [\mathbf{p}_2]_2, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle + \langle [\mathbf{p}_1]_1 \otimes [\mathbf{r}_2]_2, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle \\ &= \langle m \cdot [\mathbf{a}]_T, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle + \langle [\mathbf{r}_1]_1, \mathbf{s}_1 \rangle \bullet \langle [\mathbf{p}_2]_2, \mathbf{s}_2 \rangle + \langle [\mathbf{p}_1]_1, \mathbf{s}_1 \rangle \bullet \langle [\mathbf{r}_2]_2, \mathbf{s}_2 \rangle \\ &= \langle m \cdot [\mathbf{a}]_T, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle + \langle [\mathbf{r}_1]_1, \mathbf{s}_1 \rangle \bullet [\mathbf{0}]_2 + [\mathbf{0}]_1 \bullet \langle [\mathbf{r}_2]_2, \mathbf{s}_2 \rangle \\ &= \langle m \cdot [\mathbf{a}]_T, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle = m \cdot \langle [\mathbf{a}]_T, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle = m \cdot \langle [\mathbf{c}_{T,2}]_T, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle\end{aligned}$$

Since the encryption algorithm does not change, the security proof remains the same. The re-encryption is still possible with the following re-encryption key:

$\mathsf{REKeygen}_s(\mathsf{sk}^a_s, \mathsf{sk}^b_s)$**:** For $s = 1, 2$, from two different secret keys $\mathsf{sk}^a_s = \mathbf{s}_s$ and $\mathsf{sk}^b_s = \mathbf{s}'_s$ associated respectively to the two public keys pk^a_s and pk^b_s, take any matrix $\mathbf{M} \in \mathrm{GL}_2(\mathbb{Z}_p)$ such that $\mathsf{sk}_s = \mathsf{sk}'_s \cdot \mathbf{M}$ and output $\mathsf{rk}^{a\to b}_s = \mathbf{M}^T$ the secret re-encryption key. From $\mathsf{rk}^{a\to b}_1 \leftarrow \mathsf{REKeygen}_1(\mathsf{sk}^a_1, \mathsf{sk}^b_1)$ and $\mathsf{rk}^{a\to b}_2 \leftarrow \mathsf{REKeygen}_2(\mathsf{sk}^a_2, \mathsf{sk}^b_2)$, we denote $\mathsf{rk}^{a\to b}_T = (\mathsf{rk}^{a\to b}_1, \mathsf{rk}^{a\to b}_2)$.

As explained in the proof of Proposition 9, using equation 2: $\langle \mathbf{p}_s \cdot \mathbf{M}^T, \mathbf{s}'_s \rangle = \langle \mathbf{p}_s, \mathbf{s}'_s \cdot \mathbf{M} \rangle = \langle \mathbf{p}_s, \mathbf{s}_s \rangle = 0$, and so $\mathbf{p}_s \cdot \mathsf{rk}^{a\to b}_s = r'\mathbf{p}'_s$ for some $r' \in \mathbb{Z}_p$, which is enough to guarantee the correctness. Eventually, the distributed decryption is also possible: the user i will choose $\mathsf{rk} \overset{\$}{\leftarrow} \mathrm{GL}_2(\mathbb{Z}_p)$ and compute $\mathsf{pk}_i = \mathsf{pk}_{i-1} \cdot \mathsf{rk}^{-1}$. Its implicit secret key is $\mathsf{sk}_i = \mathsf{sk}_{i-1} \cdot \mathsf{rk}^T$.

Since the decryption key is just a vector (2 scalars), instead of a matrix (4 scalars), this also improves on the zero-knowledge proofs for the verifiability.

Acknowledgments

This work was supported in part by the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud).

References

BCFG17. Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. Cryptology ePrint Archive, Report 2017/151, 2017. `http://eprint.iacr.org/2017/151`.

BGN05. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 325–341. Springer, Heidelberg, February 2005.

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.

CDG$^+$17. Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. Cryptology ePrint Archive, Report 2017/989, 2017. `http://eprint.iacr.org/2017/989`.

CF15. Dario Catalano and Dario Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 1518–1529. ACM Press, October 2015.

DGP18. Edouard Dufour Sans, Romain Gay, and David Pointcheval. Reading in the dark: Classifying encrypted digits with functional encryption. Cryptology ePrint Archive, Report 2018/206, 2018. `https://eprint.iacr.org/2018/206`.

Dor43. R. Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943.

FLOP18. Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 331–361. Springer, Heidelberg, August 2018.

FN94. Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 480–491. Springer, Heidelberg, August 1994.

Fre10. David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 44–61. Springer, Heidelberg, May / June 2010.

Gay16. Romain Gay. Functional encryption for quadratic functions, and applications to predicate encryption. Cryptology ePrint Archive, Report 2016/1106, 2016. `http://eprint.iacr.org/2016/1106`.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

GGG$^+$14. Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 578–602. Springer, Heidelberg, May 2014.

GGH$^+$13. 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.

GGJS13. Shafi Goldwasser, Vipul Goyal, Abhishek Jain, and Amit Sahai. Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/727, 2013. `http://eprint.iacr.org/2013/727`.

GKL$^+$13. S. Dov Gordon, Jonathan Katz, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/774, 2013. `http://eprint.iacr.org/2013/774`.

GKP$^+$13. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.

LW11. Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 568–588. Springer, Heidelberg, May 2011.

NNL01. Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 41–62. Springer, Heidelberg, August 2001.

PPS12. Duong Hieu Phan, David Pointcheval, and Mario Strefler. Decentralized dynamic broadcast encryption. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 166–183. Springer, Heidelberg, September 2012.

Sch14. Rob Schapire. Computer science 511 – theoretical machine learning, 2014.

Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

SW05. Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.