Decentralized Evaluation of Quadratic Polynomials on Encrypted Data

Chloé Hébant, Duong Hieu Phan, David Pointcheval

This paper is a slight variant of the Extended Abstract that appears in
the Proceedings of the 22nd Information Security Conference, ISC 2019 (September 16–18, New-York, USA)
Springer-Verlag, LNCS ?????, pages ???–???.

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 Since the seminal paper on Fully Homomorphic Encryption (FHE) by Gentry in 2009, a lot of work and improvements have been proposed, with an amazing number of possible applications. It allows outsourcing any kind of computations on encrypted data, and thus without leaking any information to the provider who performs the computations. This is quite useful for many sensitive data (finance, medical, etc.).
Unfortunately, FHE fails at providing some computation on private inputs to a third party, in cleartext: the user that can decrypt the result is able to decrypt the inputs. A classical approach to allow limited decryption power is distributed decryption. But none of the actual FHE schemes allows distributed decryption, at least with an efficient protocol.
In this paper, we revisit the Boneh-Goh-Nissim (BGN) cryptosystem, and the Freeman's variant, that allow evaluation of quadratic polynomials, or any 2-DNF formula. Whereas the BGN scheme relies on integer factoring for the trapdoor in the composite-order group, and thus possesses one public/secret key only, the Freeman's scheme can handle multiple users with one general setup that just needs to define a pairing-based algebraic structure. We show that it can be efficiently decentralized, with an efficient distributed key generation algorithm, without any trusted dealer, but also efficient distributed decryption and distributed re-encryption, in a threshold setting. We then provide some applications of computations on encrypted data, without central authority.
Keywords: Decentralization, Fully Homormorphic Encryption, 2-DNF

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 way 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 in decentralizing computations. 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 [CC09] implies some constraints on the access control policy, that are removed in [LW11], but at the cost of the use of bilinear groups of composite order with 3 prime factors; 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] and on Functional Encryption (FE) [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]. It is of practical interest to consider the decentralization for FHE and FE without need of trust in any authority. In FE, the question in the multi-client setting was recently addressed by Chotard *et al.* [CDG+18] for the inner product function and then improved in [ABKW19, CSG+18], where all the clients agree and contribute to generate the functional decryption keys, there is no need of central authority anymore. Note that, in FE, there are efficient solutions for quadratic functions [Gay16, BCFG17] but actually, only linear function evaluations can be decentralized as none of the methods to decentralize linear schemes seems to apply, and no new method has been proposed so far. We consider, in this paper, the practical case of decentralizing FHE in multi-user setting. However, the general solution for FHE is still not yet practical, as FHE decryption requires rounding operations that are hard to efficiently distribute. Thus we only consider decentralized evaluation of quadratic polynomials. Moreover, as only decentralized linear computation was possible before, this paper can improve the efficiency of multi-party computation scheme by allowing quadratic steps and thus, improve the number of interactions needed. To motivate our work, we focus on some real-life applications in the last section which only require evaluations of quadratic polynomials so that specific target users can get the result in clear, by running re-encryption in a distributed manner under the keys of the target users.

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. Indeed, no efficient multi-party generation of distributed RSA modulus is known, except for 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. They are able to support decryption of a large plaintext space after the multiplication. However, as in Kawai *et al.* [KMH$^+$19] which used this technique to perform proxy re-encryption, 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 most of the applications and we do not try to decentralize these limited protocols.

Our Approach. Freeman [Fre10] proposed a conversion from composite-order groups to prime-order groups for the purpose of improving the efficiency. Interestingly, Freeman's conversion allows multi-user setting, since a common setup can handle several keys. But we additionally show it is well-suited for distributed evaluation of 2-DNF formulae. Actually, working in prime-order groups, we can avoid the bottleneck of a distributed generation of RSA moduli. However, it is not enough to have an efficient distributed setup. One also needs to distribute any use of the private keys in the construction: for decryption and re-encryption (see Section A.10). Unfortunately, the Freeman's generic description with projection matrices does not directly allow the design of a decentralized scheme, *i.e.*, with efficient distributed (threshold) decryption without any trusted dealer. We thus specify particular projections, with well-chosen private and public keys. This leads to an efficient decentralized version with distributed private computations. Our main contribution is to prove that using particular projection matrices does not weaken the global construction.

Related Work. In a previous and independent work, Attrapadung *et al.* [AHM$^+$18] proposed an efficient two-level homomorphic encryption in prime-order groups. They put forward a new approach that avoids the Freeman's transformation from BGN encryption. Interestingly, our work shows this scheme falls into the Freeman's framework because their construction is similar to the simplified non-decentralized version of our scheme which is obtained from BGN via a Freeman transformation with a particular choice of projections. The concrete implementations in [AHM$^+$18] show that such a scheme is quite efficient, which applies to our construction, and even to the distributed construction as each server, for a partial decryption, essentially has to perform a decryption with its share. In another unpublished work [CPRT18], Culnane *et al.*

considered a universally verifiable MPC protocol in which one of the two steps is to distribute the key generation in somewhat homomorphic cryptosystems. However, as we mentioned above, the Freeman's generic description with projection matrices, as considered in [CPRT18], does not lead to an efficient distributed decryption. In short, our result bridges the gap between the objective of decentralization as in [CPRT18] and the efficiency goal as in [AHM$^+$18].

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. However, the use of our decentralized scheme for electronic voting is much more preferable than the BGN scheme, as there is no way to trust any dealer in such a use-case. 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 these applications is highly non-trivial and will be explained in the last section.

2 Preliminaries

2.1 Notations

We denote by $x \overset{\$}{\leftarrow} X$ the process of selecting x uniformly at random in the set X. Let $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ be the ring of integers *modulo p*. For any group $\mathbb{G}$, we denote by $\langle g \rangle$ the space generated by $g \in \mathbb{G}$.

2.2 Bilinear Group Setting

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 $\mathbb{G}_1 \neq \mathbb{G}_2$. This will be our setting, while the BGN encryption scheme uses a symmetric setting, with composite-order groups.

2.3 Computational Assumption

Our security results rely on the Decisional Diffie-Hellman assumption:

Definition 1. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p. The advantage $\mathrm{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\mathcal{A})$ of an adversary $\mathcal{A}$ against the Decisional Diffie-Hellman (DDH) problem in $\mathbb{G}$ is defined by:

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

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 $\mathrm{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\mathcal{A})$ is bounded by ε.

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

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

Figure 1. Experiment of IND-CPA

2.4 Security Notions

Let us now recall the semantic security, *a.k.a.* indistinguishability (or IND-CPA), for a public-key encryption scheme, according to the experiment presented in Figure 1, where the attack is in two steps, and so the adversary outputs a state s to resume the process in the second step.

Definition 2. 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 1. 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 $\Pr[\mathrm{Exp}_{\mathcal{E}}^{\textsf{ind-cpa-1}}(\mathcal{A}) = 1] - \Pr[\mathrm{Exp}_{\mathcal{E}}^{\textsf{ind-cpa-0}}(\mathcal{A}) = 1]$. 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.

3 Encryption for Quadratic Polynomial Evaluation

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. Unfortunately, composite-order groups require huge orders, since the factorization must be difficult, with costly pairing evaluations.

3.1 Freeman's Framework

In order to improve on the efficiency, 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:

$\textsf{Keygen}(\lambda)$**:** Given a security parameter λ, 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 $g \overset{\$}{\leftarrow} G$ and $h \overset{\$}{\leftarrow} H$, and outputs the public key $\textsf{pk} = (G, H, g, h, G_1, H_1)$ and the private key $\textsf{sk} = (\pi_1, \pi_2)$.

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

$\textsf{Decrypt}(\textsf{sk}, C)$**:** 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 3. 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 Optimized Version

In the Appendix A, we present the translation of the Freeman's approach with projection matrices. This indeed leads to a public-key encryption scheme that can evaluate quadratic polynomials in $\mathbb{Z}_p$, under the DDH assumption. However, because of the secret key that must be a projection matrix, the distributed generation, while possible (see Appendix A.10), is not as efficient as one can expect. We thus now propose a particular instantiation of projections, which allows very compact keys and ciphertexts. In addition, this will allow to generate keys in a distributed manner, without any trusted dealer. While in the generic transformation of Freeman, the secret key belongs to the whole projection matrix space, our particular instantiation of projections means that the secret key will belong to a proper sub-space of the projection matrix space.

Indeed, it is possible to reduce by a factor two the size of the keys: for $s \in \{1,2\}$, the secret key is just one scalar and the public key one group element in $\mathbb{G}_s$. For the keys, we will consider orthogonal projections on $\langle(1,x)\rangle$, for any $x \in \mathbb{Z}_p$. Thus, sk_s can simply be described by $x \in \mathbb{Z}_p$, which is enough to define the projection. The public key pk_s can simply be described by $g_s^{-x} \in \mathbb{G}_s$, which is enough to define (g_s^{-x}, g_s), as $(-x,1)$ is a vector in the kernel of the projection, to add noise that the secret key will be able to remove.

More precisely, we can describe our optimized encryption schemes, for $s \in \{1,2,T\}$, as $\mathcal{E}_s : (\mathsf{Setup}, \mathsf{Keygen}_s, \mathsf{Encrypt}_s, \mathsf{Decrypt}_s)$ with a common Setup:

$\mathsf{Setup}(\lambda)$**:** Given a security parameter λ, 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 $x_s \xleftarrow{\$} \mathbb{Z}_p$ and output the public key $\mathsf{pk}_s = g_s^{-x_s}$ and the private key $\mathsf{sk}_s = x_s$. 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 $m \in \mathbb{Z}_p$ using public key pk_s, choose $r \xleftarrow{\$} \mathbb{Z}_p$ and output the ciphertext

$$C_s = (c_{s,1} = g_s^m \cdot \mathsf{pk}_s^r, c_{s,2} = g_s^r) \in \mathbb{G}_s^2.$$

For $s = T$, to encrypt a message $m \in \mathbb{Z}_p$ using public key $\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2)$, choose $r_{11}, r_{12}, r_{21}, r_{22} \xleftarrow{\$} \mathbb{Z}_p^4$ and output the ciphertext

$$C_T = \begin{pmatrix} c_{T,1} = e(g_1,g_2)^m \cdot e(g_1,\mathsf{pk}_2)^{r_{11}} \cdot e(\mathsf{pk}_1,g_2)^{r_{21}}, \\ c_{T,2} = e(g_1,g_2)^{r_{11}} \cdot e(\mathsf{pk}_1,g_2)^{r_{22}}, \\ c_{T,3} = e(g_1,\mathsf{pk}_2)^{r_{12}} \cdot e(g_1,g_2)^{r_{21}}, \\ c_{T,4} = e(g_1,g_2)^{r_{12}+r_{22}} \end{pmatrix} \in \mathbb{G}_T^4$$

$\mathsf{Decrypt}_s(\mathsf{sk}_s, C_s)$**:** For $s \in \{1,2\}$, given $C_s = (c_{s,1}, c_{s,2})$ and the private key sk_s, compute $d = c_{s,1} \cdot c_{s,2}^{\mathsf{sk}_s}$ and output the logarithm of d in basis g_s. For $s = T$, given $C_T = (c_{T,1}, c_{T,2}, c_{T,3}, c_{T,4})$ and $\mathsf{sk}_T = (\mathsf{sk}_1, \mathsf{sk}_2)$, compute $d = c_{T,1} \cdot c_{T,2}^{\mathsf{sk}_2} \cdot c_{T,3}^{\mathsf{sk}_1} \cdot c_{T,4}^{\mathsf{sk}_1 \cdot \mathsf{sk}_2}$ and output the logarithm of d in basis $e(g_1, g_2)$.

In $\mathbb{G}_1$ and $\mathbb{G}_2$, this is actually the classical ElGamal encryption. We essentially extend it to $\mathbb{G}_T$, to handle quadratic operations:

$\mathsf{Add}(C_s, C'_s)$ just consists of the component-wise product in $\mathbb{G}_s$;

$\mathsf{Multiply}(C_1, C_2)$ for $C_1 = (c_{1,1} = g_1^{m_1} \cdot \mathsf{pk}_1^{r_1}, c_{1,2} = g_1^{r_1}) \in \mathbb{G}_1^2$ and $C_2 = (c_{2,1} = g_2^{m_2} \cdot \mathsf{pk}_2^{r_2}, c_{2,2} = g_2^{r_2}) \in \mathbb{G}_2^2$, consists of the tensor product:

$$C_T = (e(c_{1,1}, c_{2,1}), e(c_{1,1}, c_{2,2}), e(c_{1,2}, c_{2,1}), e(c_{1,2}, c_{2,2})) \in \mathbb{G}_T^4$$

$\mathsf{Randomize}_s(\mathsf{pk}_s, C_s)$ is, as usual, the addition of a random ciphertext of 0 in the same group $\mathbb{G}_s$. For $s \in \{1,2\}$: Given a ciphertext $C_s = (c_{s,1}, c_{s,2})$ with its public key pk_s, it chooses $r \xleftarrow{\$} \mathbb{Z}_p$ and outputs $(c_{s,1} \cdot \mathsf{pk}_s^r, c_{s,2} \cdot g_s^r)$; while for $s = T$, a public key pk_T and a ciphertext $(c_{T,1}, c_{T,2}, c_{T,3}, c_{T,4})$, it chooses $r'_{11}, r'_{12}, r'_{21}, r'_{22} \xleftarrow{\$} \mathbb{Z}_p$ and outputs $(c_{T,1} \cdot e(g_1, \mathsf{pk}_2)^{r'_{11}} \cdot e(\mathsf{pk}_1, g_2)^{r'_{21}}, c_{T,2} \cdot e(g_1, g_2)^{r'_{11}} \cdot e(\mathsf{pk}_1, g_2)^{r'_{22}}, c_{T,3} \cdot e(g_1, \mathsf{pk}_2)^{r'_{12}} \cdot e(g_1, g_2)^{r'_{21}}, c_{T,4} \cdot e(g_1, g_2)^{r'_{12}+r'_{22}})$.

3.3 Security Properties

Whereas the correctness directly comes from the correctness of the Freeman's construction, presented in the Appendix A, and verification is straightforward, the semantic security comes from the classical ElGamal encryption security, under the DDH assumptions, for the basic schemes in $\mathbb{G}_1$ and $\mathbb{G}_2$:

Theorem 4. *For $s \in \{1,2\}$, $\mathcal{E}_s$ is* IND-CPA *under the* DDH *assumption in $\mathbb{G}_s$: for any adversary $\mathcal{A}$ running within time t, $\mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_s}(\mathcal{A}) \leq 2 \times \mathrm{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t)$.*

Corollary 5. *$\mathcal{E}_T$ is* IND-CPA *under the* DDH *assumptions in $\mathbb{G}_1$ or $\mathbb{G}_2$.*

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

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

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 IND-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 $b = 0$:
 - $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{Setup}(\lambda)$
 - $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{Keygen}_1(\mathsf{param})$,$(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{Keygen}_2(\mathsf{param})$
 - $m_0, m_1 \leftarrow \mathcal{A}(\mathsf{param}, (\mathsf{pk}_1, \mathsf{pk}_2))$; $C_T = \mathsf{Encrypt}_T((\mathsf{pk}_1, \mathsf{pk}_2), m_0)$
 - $\beta \leftarrow \mathcal{A}(\mathsf{param}, (\mathsf{pk}_1, \mathsf{pk}_2), C_T)$

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 $\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_1}(\mathcal{A})$, 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) the simulators forwards to the challenger. It gets back $C_1 = \mathsf{Encrypt}_1(\mathsf{pk}_1, m_0)$. It can compute $C_T = \mathsf{Multiply}(C_1, \mathsf{Encrypt}_2(\mathsf{pk}_2, 1))$, 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 $\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_1}(\mathcal{A})$, where $b = 1$:

$$\Pr_{\mathbf{G}_2}[E] - \Pr_{\mathbf{G}_1}[E] \leq \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_1}(t + 4 \cdot t_p + 4 \cdot t_e),$$

where t_p is the time for one pairing and t_e the time for one exponentiation.

Game $\mathbf{G}_3$: In this final game, the simulator plays the role of the challenger in $\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_T}(\mathcal{A})$, 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, $\Pr_{\mathbf{G}_3}[E] = \Pr\left[\mathrm{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_T}(\mathcal{A}) = 1\right]$, hence

$$\mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_T}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_1}(t + 4 \cdot t_p + 4 \cdot t_e),$$

which concludes the proof, since it works exactly the same way for $\mathbb{G}_2$. □

We stress that the security of $\mathcal{E}_T$ only requires the DDH assumption in one of the two groups, and not the SXDH assumption (which means that the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$).

4 Decentralized Homomorphic Encryption

Our main motivation was a decentralized key generation and a distributed decryption in order to be able to compute on encrypted data so that nobody can decrypt intermediate values but the result can be provided in clear to a target user. We now show that our optimized construction allows both decentralized key generation without a trusted dealer and distributed decryption. They are both quite efficient. When a result should be available to a unique user, a classical technique called proxy re-encryption [BBS98] is to re-encrypt to this target user: this is a virtual decryption followed by encryption under the new key. We also show this is possible to do it in a distributed way, without any leakage of information.

4.1 Decentralized Key Generation

In fact, a classical decentralized t-out-of-n threshold secret sharing allows to generate the shares of a random element and it seems hard (if one expects efficiency) to use it to generate the shares of a structured matrix, such as projections required in the generic construction, because its elements are not independently random. In our specific construction, the secret keys in $\mathbb{G}_1$ and $\mathbb{G}_2$ are now one scalar and one can perform a classical t-out-of-n threshold secret sharing: each player i generates a random polynomial P_i of degree $t-1$ in $\mathbb{Z}_p[X]$, privately sends $x_{i,j} = P_i(j)$ to player j, and publishes $g_s^{-P_i(0)}$; each player i then aggregates the values into $\mathsf{sk}_i = \sum_j x_{j,i} = P(i)$, for $P = \sum_j P_j$, which leads to a share of $x = P(0)$, and the public key is the product of all the public values.

4.2 Distributed Decryption

In order to decrypt $C_s = (c_{s,1}, c_{s,2})$ in $\mathbb{G}_1$ or $\mathbb{G}_2$, each player in a sub-set of t players sends its contribution $c_{s,2}^{\mathsf{sk}_i}$, that can be multiplied with Lagrange coefficients as exponents to obtain the mask $c_{s,2}^{\mathsf{sk}} = \mathsf{pk}_s^{-r}$. To decrypt $C_T = (c_{T,1}, c_{T,2}, c_{T,3}, c_{T,4})$ in $\mathbb{G}_T$, one can first use the shares of sk_1 to compute $c_{T,3}^{\mathsf{sk}_1}$ and $c_{T,4}^{\mathsf{sk}_1}$, and then the shares of sk_2 to compute $c_{T,2}^{\mathsf{sk}_2}$ and $c_{T,4}^{\mathsf{sk}_1 \cdot \mathsf{sk}_2}$. Under the DDH assumptions in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, one can show that the intermediate values $c_{s,2}^{\mathsf{sk}_i}$, or $c_{T,3}^{\mathsf{sk}_1}$, $c_{T,4}^{\mathsf{sk}_1}$, $c_{T,2}^{\mathsf{sk}_2}$, and $c_{T,4}^{\mathsf{sk}_1 \cdot \mathsf{sk}_2}$ do not leak more than the decryption itself. Of course, classical verifiable secret sharing techniques can be used, for both the decentralized generation and the distributed decryption. This can allow, with simple Schnorr-like proofs of Diffie-Hellman tuples, universal verifiability.

4.3 Distributed Re-encryption

Besides a distributed decryption, when outsourcing some computations on private information, a distributed authority may want to re-encrypt the encrypted result to a specific user, so that the latter can get the result in clear, and nobody else. More precisely, we assume the input data were encrypted under the keys pk_1, pk_2, and $\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2)$, which leads, after quadratic evaluations, to a resulting ciphertext under the key pk_T, for which the distributed authorities, knowing a t-out-of-n additive secret sharing $(\mathsf{sk}_{1,i}, \mathsf{sk}_{2,i})_i$ of $(\mathsf{sk}_1, \mathsf{sk}_2)$, will re-encrypt under $\mathsf{PK}_T = (\mathsf{PK}_1, \mathsf{PK}_2)$ for the target user.

Of course, such a re-encryption can be performed using multi-party computation, but we will show an efficient way to do it. And we start with the re-encryption of $c_s = (c_{s,1} = g_s^m \cdot \mathsf{pk}_s^r, c_{s,2} = g_s^r)$: player i chooses $r'_i \xleftarrow{\$} \mathbb{Z}_p$, computes $\alpha_i = c_{s,2}^{\mathsf{sk}_{s,i}} \cdot \mathsf{PK}_s^{r'_i}$ and $\beta_i = g_s^{r'_i}$, and outputs (α_i, β_i). Then, anybody can compute, for the appropriate Lagrange coefficients λ_i's,

$$C_s = (C_{s,1} = c_{s,1} \times \prod \alpha_i^{\lambda_i} = g_s^m \mathsf{pk}_s^r g_s^{r\cdot\mathsf{sk}_s} \cdot \mathsf{PK}_s^{r'} = g_s^m \cdot \mathsf{PK}_s^{r'}, C_{s,2} = \prod \beta_i^{\lambda_i} = g_s^{r'})$$

with $r' = \sum \lambda_i r'_i$, where the sum is on the t members available.

For $s = T$, given a ciphertext $c_T = (c_{T,1}, c_{T,2}, c_{T,3}, c_{T,4})$, player i chooses $u_i \xleftarrow{\$} \mathbb{Z}_p$, and first computes and sends $\alpha_{3,i} = c_{T,4}^{\mathsf{sk}_{1,i}} \cdot e(g_1, g_2)^{-u_i}$. With a linear combination for the appropriate Lagrange coefficients λ_i's, anybody can compute, $\alpha_3 = \prod \alpha_{3,i}^{\lambda_i} = c_{T,4}^{\mathsf{sk}_1} \cdot e(g_1, g_2)^{-u}$, with implicit $u = \sum \lambda_i u_i$. Then each player i chooses $r'_{11,i}, r'_{12,i}, r'_{21,i}, r'_{22,i}, v_i \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$\begin{aligned}
\alpha_{1,i} &= c_{T,2}^{\mathsf{sk}_{2,i}} \cdot e(\mathsf{PK}_1, g_2)^{r'_{21,i}} & \beta_i &= e(g_1, g_2)^{r'_{11,i}+u_i} \cdot e(\mathsf{PK}_1, g_2)^{r'_{22,i}} \\
\alpha_{2,i} &= c_{T,3}^{\mathsf{sk}_{1,i}} \cdot e(g_1, \mathsf{PK}_2)^{r'_{11,i}} & \gamma_i &= e(g_1, \mathsf{PK}_2)^{r'_{12,i}} \cdot e(g_1, g_2)^{r'_{21,i}+v_i} \\
\alpha_{4,i} &= \alpha_3^{\mathsf{sk}_{2,i}} \cdot e(\mathsf{PK}_1, g_2)^{v_i} & \delta_i &= e(g_1, g_2)^{r'_{12,i}+r'_{22,i}}
\end{aligned}$$

Again, with linear combinations for the appropriate Lagrange coefficients λ_i's, anybody can compute, with $r'_{jk} = \sum \lambda_i r'_{jk,i}$, for $j, k \in \{1, 2\}$, and $v = \sum \lambda_i v_i$:

$$\begin{aligned}
\alpha_1 &= c_{T,2}^{\mathsf{sk}_2} \cdot e(\mathsf{PK}_1, g_2)^{r'_{21}} & C_{T,2} &= e(g_1, g_2)^{r'_{11}+u} \cdot e(\mathsf{PK}_1, g_2)^{r'_{22}} \\
\alpha_2 &= c_{T,3}^{\mathsf{sk}_1} \cdot e(g_1, \mathsf{PK}_2)^{r'_{11}} & C_{T,3} &= e(g_1, \mathsf{PK}_2)^{r'_{12}} \cdot e(g_1, g_2)^{r'_{21}+v} \\
\alpha_4 &= c_{T,4}^{\mathsf{sk}_1\mathsf{sk}_2} \cdot e(g_1, \mathsf{PK}_2)^u \cdot e(\mathsf{PK}_1, g_2)^v & C_{T,4} &= e(g_1, g_2)^{r'_{12}+r'_{22}}
\end{aligned}$$

Then, $C_{T,1} = c_{T,1} \times \alpha_1\alpha_2\alpha_4 = e(g_1, g_2)^m \cdot e(g_1, \mathsf{PK}_2)^{r'_{11}+u} \cdot e(\mathsf{PK}_1, g_2)^{r'_{21}+v}$, so that $C_T = (C_{T,1}, C_{T,2}, C_{T,3}, C_{T,4})$ is a re-encryption of c_T under PK_T.

For random scalars, the re-encryption algorithms (which is just a one-round protocol in $\mathbb{G}_1$ and $\mathbb{G}^2$, but 2-round in $\mathbb{G}_T$) generate new ciphertexts under appropriate keys that look perfectly fresh. In addition, one can claim:

Theorem 6. *The above distributed protocols for re-encryption do not leak additional information than the outputs of the non-distributed algorithms.*

Proof. The goal of this proof is to show that the distributed protocol to re-encrypt a ciphertext under PK_s does not leak more information than a direct encryption under PK_s. For $s \in \{1, 2\}$, one is given $c_s = \mathsf{Encrypt}_s(m, \mathsf{pk}_s; r) = (c_{s,1}, c_{s,2})$ and $C_s = \mathsf{Encrypt}_s(m, \mathsf{PK}_s; R) = (C_{s,1}, C_{s,2})$, two ciphertexts of the same message m under pk_s and PK_s respectively. One can then note that $C_{s,1}/c_{s,1} = \mathsf{PK}_s^R/\mathsf{pk}_s^r = c_{s,2}^{\mathsf{sk}_s}/C_{s,2}^{\mathsf{SK}_s}$.

The Re-Encryption in $\mathbb{G}_s$, for $s \in \{1, 2\}$.

Game $\mathbf{G}_0$: In the first game, the simulator just receives $c_s = (c_{s,1}, c_{s,2})$, and plays the real protocol using the t-out-of-n distributed keys $(\mathsf{sk}_{s,i})_i$ to provide the keys to the corrupted users and to generate the values $\alpha_i = c_{s,2}^{\mathsf{sk}_{s,i}} \cdot \mathsf{PK}_s^{r'_i}$ and $\beta_i = g_s^{r'_i}$, on behalf of the non-corrupted players. We assume that among t players, ℓ are honest and $t - \ell$ are corrupted. The latter are assumed to receive the secret keys $\mathsf{sk}_{s,i}$ and to generate their own outputs (α_i, β_i). The view of the attacker consists of the set of all the honest (α_i, β_i).

Game $\mathbf{G}_1$: The simulator is now given $c_s = (c_{s,1}, c_{s,2})$ and $C_s = (C_{s,1}, C_{s,2})$ that encrypt the same message. We want, for the appropriate Lagrange coefficients λ_i

$$C_{s,1} = c_{s,1} \cdot \prod \alpha_i^{\lambda_i} \qquad C_{s,2} = \prod \beta_i^{\lambda_i}.$$

Hence, the simulator can take, for all the honest players except the last one, $r'_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ to compute $\alpha_i = c_{s,2}^{\mathsf{sk}_{s,i}} \cdot \mathsf{PK}_s^{r'_i}$ and $\beta_i = g_s^{r'_i}$. For the last honest player, from all the honest-user shares and corrupted-user shares, one sets

$$\alpha_\ell = (C_{s,1}/c_{s,1} \cdot \prod_{i \neq \ell} \alpha_i^{-\lambda_i})^{1/\lambda_\ell} \quad \beta_\ell = (C_{s,2} \cdot \prod_{i \neq \ell} \beta_i^{-\lambda_i})^{1/\lambda_\ell}.$$

Then, for the t players: $\prod \alpha_i^{\lambda_i} = c_{s,2}^{\mathsf{sk}_s} \cdot \mathsf{PK}_s^{r'}$ and $\prod \beta_i^{\lambda_i} = g_s^{r'}$, for $r' = \sum \lambda_i r'_i$ and with the implicit $r'_\ell = (R - \sum_{i \neq \ell} \lambda_i r'_i)/\lambda_\ell$. So $r' = R$. The view of the attacker remains exactly the same.

Game $\mathbf{G}_2$: In this game, the simulator also takes as input a Diffie-Hellman tuple $(A = g^r, B = \mathsf{PK}_s^r)$ with (g_s, PK_s): it first derives enough independent pairs $(A_i, B_i) = (g_s^{x_i} \cdot A^{y_i}, \mathsf{PK}_s^{x_i} \cdot B^{y_i})$, for random x_i, y_i, for all the non-corrupted players (excepted the last one), and computes $\alpha_i = c_{s,2}^{\mathsf{sk}_{s,i}} \cdot B_i, \beta_i = A_i$. Since $(g_s, \mathsf{PK}_s, A, B)$ is a Diffie-Hellman tuple, the view is perfectly indistinguishable from the previous one.

Game $\mathbf{G}_3$: In this game, the simulator now receives a random tuple (A, B), which makes all the (A_i, B_i) independent random pairs, the rest is unchanged: under the DDH assumption in $\mathbb{G}_s$, the view is computationally indistinguishable.

Game $\mathbf{G}_4$: This is the final simulation, where all the honest shares (α_i, β_i) are chosen at random, except the last ones $(\alpha_\ell, \beta_\ell)$ that are still computed as above to complete the values using c_s and C_s: the view is perfectly indistinguishable from the previous one and does not leak information.

As a consequence, we have proven that there is a simulator (defined in the last game) that produces a view indistinguishable from the real view, with just the input-output pairs. This proves that nothing else leaks. □

The Re-Encryption in $\mathbb{G}_T$. The proof follows the same path as in the previous proof: one is given two ciphertexts $c_T = \mathsf{Encrypt}_T(m, (\mathsf{pk}_1, \mathsf{pk}_2); r_{11}, r_{12}, r_{21}, r_{22})$ and $C_T = \mathsf{Encrypt}_T(m, (\mathsf{PK}_1, \mathsf{PK}_2); R_{11}, R_{12}, R_{21}, R_{22})$ of the same message m under pk_T and PK_T respectively. One needs to simulate all the $\alpha_{1,i}, \alpha_{2,i}, \alpha_{3,i}, \alpha_{4,i}, \beta_i, \gamma_i, \delta_i$ for all the non-corrupted players. Since c_T and C_T encrypt the same message, and we want

$$C_{T,1} = c_{T,1} \cdot \prod \alpha_{1,i}^{\lambda_i} \cdot \alpha_{2,i}^{\lambda_i} \cdot \alpha_{4,i}^{\lambda_i} \qquad C_{T,2} = \prod \beta_i^{\lambda_i} \qquad C_{T,3} = \prod \gamma_i^{\lambda_i} \qquad C_{T,4} = \prod \delta_i^{\lambda_i}$$

the simulator can take, for all the honest players except the last one, $r'_{11,i}, r'_{12,i}, r'_{21,i}, r'_{22,i}, u_i, v_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ to compute, in the first round:

$$\alpha_{3,i} = c_{T,4}^{\mathsf{sk}_{1,i}} \cdot e(g_1, g_2)^{-u_i} \qquad \alpha_{3,\ell} \stackrel{\$}{\leftarrow} \mathbb{G}_T \qquad \alpha_3 = \prod \alpha_{3,i}^{\lambda_i}$$

and in the second round, for all but the last honest player

$$\begin{aligned} \alpha_{1,i} &= c_{T,2}^{\mathsf{sk}_{2,i}} \cdot e(\mathsf{PK}_1, g_2)^{r'_{21,i}} & \beta_i &= e(g_1, g_2)^{r'_{11,i} + u_i} \cdot e(\mathsf{PK}_1, g_2)^{r'_{22,i}} \\ \alpha_{2,i} &= c_{T,3}^{\mathsf{sk}_{1,i}} \cdot e(g_1, \mathsf{PK}_2)^{r'_{11,i}} & \gamma_i &= e(g_1, \mathsf{PK}_2)^{r'_{12,i}} \cdot e(g_1, g_2)^{r'_{21,i} + v_i} \\ \alpha_{4,i} &= \alpha_3^{\mathsf{sk}_{2,i}} \cdot e(\mathsf{PK}_1, g_2)^{v_i} & \delta_i &= e(g_1, g_2)^{r'_{12,i} + r'_{22,i}} \end{aligned}$$

and for the last honest player:

$$\begin{aligned}
\alpha_{2,\ell} &\xleftarrow{\$} \mathbb{G}_T & \beta_\ell &= (C_{T,2} \times \prod_{i\neq\ell} \beta_i^{-\lambda_i})^{1/\lambda_\ell} \\
\alpha_{4,\ell} &\xleftarrow{\$} \mathbb{G}_T & \gamma_\ell &= (C_{T,3} \times \prod_{i\neq\ell} \gamma_i^{-\lambda_i})^{1/\lambda_\ell} \\
 & & \delta_\ell &= (C_{T,4} \times \prod_{i\neq\ell} \delta_i^{-\lambda_i})^{1/\lambda_\ell}
\end{aligned}$$

which implies implicit values for $r'_{11,\ell}, r'_{12,\ell}, r'_{21,\ell}, r'_{22,\ell}, u_\ell, v_\ell$ because the above system is invertible, where X, Y, and Z are the constant values introduced by $c_{T,i}^{\mathsf{sk}_j}$, for some i, j:

$$\lambda_\ell \times \begin{pmatrix} \log \beta_\ell \\ \log \gamma_\ell \\ \log \delta_\ell \\ \log \alpha_{4,\ell} \\ \log \alpha_{3,\ell} \\ \log \alpha_{2,\ell} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 & -\mathsf{sk}_1 & 1 & 0 \\ 0 & \mathsf{sk}_2 & 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathsf{sk}_{2,\ell} & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} r'_{11,\ell} \\ r'_{12,\ell} \\ r'_{21,\ell} \\ r'_{22,\ell} \\ u_\ell \\ v_\ell \end{pmatrix}$$

Then it is possible to set: $\alpha_{1,\ell} = (C_{T,1}/(c_{T,1} \cdot \alpha_2\alpha_4) \times \prod_{i\neq\ell} \alpha_{1,i}^{-\lambda_i})^{1/\lambda_\ell}$.

First, this is clear that the $\alpha_{3,i}$'s do not leak anything as they contain random masks $e(g_1, g_2)^{-u_i}$. Then, to prove that all the $\alpha_{1,i}, \alpha_{2,i}, \alpha_{4,i}, \beta_i, \gamma_i, \delta_i$ do not leak information, one can perform a similar proof as above for $\mathbb{G}_s$, by using the DDH assumption in both $\mathbb{G}_1$ and $\mathbb{G}_2$. Indeed, each element is masked using a pair either (g_2^r, PK_2^r) or (g_1^r, PK_1^r), for some random r. If one wants to have an indistinguishability under the SXDH assumption (and thus only one DDH assumption in one group), one could add more masks. But this does not make sense to have one key compromised and not the other one, for the same user. Hence, we tried to make the re-encryption as efficient as possible. □

We stress that for the re-encryption in $\mathbb{G}_1$ or $\mathbb{G}_2$, one just needs the DDH assumption in this group $\mathbb{G}_s$. But for the re-encryption in $\mathbb{G}_T$, one needs the DDH assumption in both $\mathbb{G}_1$ and $\mathbb{G}_2$ (the so-called SXDH assumption). We could rely on only one of the two, by adding masking factors, but this does not really make sense for a user to have his private key sk_1 being compromised without sk_2 (or the opposite).

In addition, zero-knowledge proofs can be provided to guarantee the re-encryption is honestly applied: they just consist in proofs of representations, when $g_s^{\mathsf{sk}_{s,i}}$ are all made public, for $s \in \{1, 2\}$ and all indices i.

4.4 Efficiency

In the concrete case where we have n servers able to perform a distributed protocol as described above, each of them has two scalars corresponding to a secret key for the encryption in $\mathbb{G}_1$ and a secret key for the encryption in $\mathbb{G}_2$. We recall that a ciphertext, in $\mathbb{G}_1$ or $\mathbb{G}_2$, is composed of two group elements, and a ciphertext in $\mathbb{G}_T$ is composed of four group elements. A recipient, that wants the result of either a decryption or a re-encryption with the help of t servers, has to perform a few exponentiations. The table below details the number of exponentiations for each player involved in the distributed protocols.

5 Applications

5.1 Encryption for Boolean Formulae

In this part, we detail the specific case of the evaluation of 2-DNF.

	per server	recipient
distributed decryption in $\mathbb{G}_1/\mathbb{G}_2$	1	t
in $\mathbb{G}_T$	4	$4t$
distributed re-encryption in $\mathbb{G}_1/\mathbb{G}_2$	3	t
in $\mathbb{G}_T$	13	$7t$

First, as explained in [BGN05], a way to guarantee the ciphertexts are encryption of inputs in $\{0,1\}$, the verification can be done with our scheme (or the one of BGN or Freeman) with the additional term $\mathsf{Add}_j(\mathsf{Multiply}(C_{x_j}, \mathsf{Add}(C_{x_j}, C_{-1}))$, multiplied by a random constant, so that it adds zero if inputs are correct, or it adds a random value otherwise. This introduces a quadratic term, just for the verification. This is at no extra cost if the Boolean formula is already quadratic, which will be the case of our applications.

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 $x_1, \dots, x_n \in \{0,1\}$ is of the form

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

The conversion of 2-DNF formulae into multivariate polynomials of total degree 2 is simple: given $\Phi(x_1, \dots, x_n) = \bigvee_{i=1}^{m} (\ell_{i,1} \wedge \ell_{i,2})$ a 2-DNF formula, define $\phi(x_1, \dots, x_n) = \sum_{i=1}^{m} (y_{i,1} \times y_{i,2})$ where $y_{i,j} = \ell_{i,j}$ if $\ell_{i,j} \in \{x_1, \dots, x_n\}$ 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 $x \in \{0,1\}$. $\phi(x_1, \dots, x_n)$ is the multivariate polynomial of degree 2 corresponding to $\Phi(x_1, \dots, x_n)$. 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.

Because one just wants to know whether the result is equal to 0 or the result is different from 0, we do not need to compute the logarithm: we can decrypt by just checking whether $c_{s,1} \cdot c_{s,2}^{\mathsf{sk}_s} = 1_s$ or not (for $s = T$, if $c_{T,1} \cdot c_{T,2}^{\mathsf{sk}_2} \cdot c_{T,3}^{\mathsf{sk}_1} \cdot c_{T,4}^{\mathsf{sk}_1 \cdot \mathsf{sk}_2} = 1_T$).

5.2 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 evaluate is $\neg\mathsf{F}_i(\mathbf{X}, \mathbf{y})$, which means the patient's test is positive (infected) or not, for $\mathsf{F}_i(\mathbf{X}, \mathbf{y}) = \bigvee_j (x_{ij} \wedge \neg y_j)$. The latter is indeed true if there is a mix containing a i-th sample for which the test is negative, and this should claim patient i 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, $C_{\overline{y_j}} \in \mathbb{G}_2^2$, the encryption of $\overline{y_j}$. Then, they both send the ciphertexts to an external database. With our homomorphic encryption scheme, to compute $\neg\mathsf{F}_i$, we can publicly evaluate the following formula: $C_i = \mathsf{Randomize}(\mathsf{Add}_j(\mathsf{Multiply}(C_{x_{ij}}, C_{\overline{y_j}}))$. Anybody can publicly verify the computations and if it is correct, a pool of controllers perform a distributed re-encryption of the result of patient i under his key PK_i. In this way, the patient cannot decrypt the database or the result of the tests directly, but only with the help of a pool of controller. The goal of the controllers is to limit access to the specific users only. Under an assumption about the collusions among the controllers, nobody excepted the users will have access to their own results.

5.3 Consistency Model on Encrypted Data

Another famous application 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 pool of distributed controllers to limit decryptions, but the controllers should not learn anything either.

Suppose 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 whether 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 viewed 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 if 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 we want to compute on the encrypted inputs:

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

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 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

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

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 controllers can compute:

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

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 pool of controllers re-encrypt for the user in a distributed way.

Acknowledgments

This work was supported in part by the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud) and the French ANR ALAMBIC Project (ANR16-CE39-0006).

References

ABKW19. Michel Abdalla, Fabrice Benhamouda, Markulf Kolhweiss, and Hendrik Waldner. Decentralizing inner-product functional encryption. Cryptology ePrint Archive, Report 2019/020, 2019. `https://eprint.iacr.org/2019/020`.

AHM$^+$18. Nuttapong Attrapadung, Goichiro Hanaoka, Shigeo Mitsunari, Yusuke Sakai, Kana Shimizu, and Tadanori Teruya. Efficient two-level homomorphic encryption in prime-order bilinear groups and A fast implementation in WebAssembly. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *ASIACCS 18*, pages 685–697. ACM Press, April 2018.

BBS98. Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 127–144. Springer, Heidelberg, May / June 1998.

BCFG17. Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 67–98. Springer, Heidelberg, August 2017.

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.

CC09. Melissa Chase and Sherman S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *ACM CCS 09*, pages 121–130. ACM Press, November 2009.

CDG$^+$18. Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 703–732. Springer, Heidelberg, December 2018.

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.

CPRT18. Chris Culnane, Olivier Pereira, Kim Ramchen, and Vanessa Teague. Universally verifiable MPC with applications to IRV ballot counting. Cryptology ePrint Archive, Report 2018/246, 2018. `https://eprint.iacr.org/2018/246`.

CSG$^+$18. Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Multi-client functional encryption with repetition for inner product. Cryptology ePrint Archive, Report 2018/1021, 2018. `https://eprint.iacr.org/2018/1021`.

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.

KMH$^+$19. Yutaka Kawai, Takahiro Matsuda, Takato Hirano, Yoshihiro Koseki, and Goichiro Hanaoka. Proxy re-encryption that supports homomorphic operations for re-encrypted ciphertexts. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E102.A:81–98, 01 2019.

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.

– For $x \in \mathbb{Z}_p, \mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p)$: $[x] = g^x$, $[\mathbf{A}] = g^{\mathbf{A}} = (g^{a_{ij}})_{ij}$

– For $x \in \mathbb{Z}_p, \mathbf{A}, \mathbf{B} \in \mathcal{M}_{m,n}(\mathbb{Z}_p), \mathbf{X} \in \mathcal{M}_{n,n'}(\mathbb{Z}_p), \mathbf{Y} \in \mathcal{M}_{m',m}(\mathbb{Z}_p)$:

$$x \cdot [\mathbf{A}] = g^{x\mathbf{A}} \qquad [\mathbf{A}] \cdot \mathbf{X} = g^{\mathbf{AX}} \qquad \mathbf{Y} \cdot [\mathbf{A}] = g^{\mathbf{YA}} \qquad [\mathbf{A}] + [\mathbf{B}] = [\mathbf{A} + \mathbf{B}]$$

– For $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p)$, $\mathbf{B} \in \mathcal{M}_{m',n'}(\mathbb{Z}_p)$: $[\mathbf{A}]_1 \bullet [\mathbf{B}]_2 = [\mathbf{A} \otimes \mathbf{B}]_T$

Figure 2. Bracket Notations

A Freeman's Approach

A.1 Notations

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 $\mathbf{x} \overset{\$}{\leftarrow} X^n$, $\mathbf{x} = (x_1 \ x_2 \ \cdots \ x_n) = (x_1, x_2, \cdots, x_n)$.

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 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 $\mathbf{B} \in \mathcal{M}_{m',n'}(\mathbb{Z}_p)$,

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

The bilinearity of the tensor product gives, 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 details about the "brackets". In case of bilinear groups, 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.

A.2 Projections

In order to continue with matrix properties and linear applications, a *projection* π, in a space of dimension n, is a linear function such that $\pi \circ \pi = \pi$. Any projection of rank 1 can be represented by the matrix $\mathbf{P} = \mathbf{B}^{-1}\mathbf{U}_n\mathbf{B}$, where $\mathbf{U}_n$ is the canonical projection and $\mathbf{B}$ is the change of basis matrix:

$$\mathbf{U}_n = \begin{pmatrix} 0 \ldots 0 \ 0 \\ 0 \ \ddots \ 0 \ 0 \\ 0 \ldots 0 \ 0 \\ 0 \ldots 0 \ 1 \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_{n-1} \\ \mathbf{b} \end{pmatrix}$$

where $K = \langle \mathbf{p}_1, \ldots, \mathbf{p}_{n-1} \rangle$ is the kernel of the projection and $\langle \mathbf{b} \rangle$ the image.

Given two projections π_1 and π_2 of rank 1, that 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

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

The associated change of basis matrix is thus $\mathbf{B} = \mathbf{B}_1 \otimes \mathbf{B}_2$. In 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. But, as explained below, $\mathbf{p}_1 \otimes \mathbf{r}_2 + \mathbf{r}_1 \otimes \mathbf{p}_2$, for $\mathbf{r}_1, \mathbf{r}_2 \overset{\$}{\leftarrow} \mathbb{Z}_p^2$, provides a uniform sampling in $\ker(\pi)$:

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

A.3 Freeman's Scheme with Projections

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.

We now present a variant of the Freeman's cryptosystem allowing multiple users, 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 $\mathsf{Keygen}, \mathsf{Encrypt}$ and $\mathsf{Decrypt}$ will take a sub-script s to precise the group $\mathbb{G}_s$ in which they operate, but the Setup is common.

$\mathsf{Setup}(\lambda)$**:** Given a security parameter λ, 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 $\mathbf{B}_s \overset{\$}{\leftarrow} \mathrm{GL}_2(\mathbb{Z}_p)$, let $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}_2\mathbf{B}_s$ and $\mathbf{p}_s \in \ker(\mathbf{P}_s) \setminus \{\mathbf{0}\}$, and 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 $m \in \mathbb{Z}_p$ 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 masks the message by an element in the kernel of a certain projection. The secret key is the corresponding projection $\mathbf{P}_s$ which later removes this mask. In the Decrypt algorithm, C'_s is a Diffie-Hellman tuple (whatever the group under consideration), the discrete logarithm of one component is enough 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.

A.4 Correctness of $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_T$

Proposition 7. *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. $\square$

A.5 Homomorphic Properties

As BGN, Freeman cryptosystem also allows additions, one multiplication layer, and additions: 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 $C_T = ([\mathbf{c}_{1,1}]_1 \bullet [\mathbf{c}_{2,1}]_2, [\mathbf{c}_{1,2}]_1 \bullet [\mathbf{c}_{2,2}]_2) \in \mathbb{G}_T^4 \times \mathbb{G}_T^4$.

$\mathsf{Randomize}_s(\mathsf{pk}_s, C_s)$**:** 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 $\alpha, r \overset{\$}{\leftarrow} \mathbb{Z}_p$ 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 $\alpha \overset{\$}{\leftarrow} \mathbb{Z}_p, [\mathbf{r}_1]_1 \overset{\$}{\leftarrow} \mathbb{G}_1^2$ and $[\mathbf{r}_2]_2 \overset{\$}{\leftarrow} \mathbb{G}_2^2$, 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 randomization of ciphertexts as proposed by Freeman each time an Add or a Multiply is computed, we create a specific function Randomize usable at any time, when more privacy is required.

A.6 Correctness of the Homomorphic Properties

Let us check the correctness of the three homomorphic functions:

Proposition 8. 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 $s = T$:

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

About multiplication, we can see that

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

□

Proposition 9. *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 $s = T$:

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

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 $\mathbb{G}_2$. □

A.7 Security Properties

Theorem 10. *For $s \in \{1,2\}$, $\mathcal{E}_s$ is* IND-CPA *under the* DDH *assumption in $\mathbb{G}_s$: for any adversary $\mathcal{A}$ running within time t, $\mathrm{Adv}_{\mathcal{E}_s}^{\mathsf{ind\text{-}cpa}}(\mathcal{A}) \leq 2 \times \mathrm{Adv}_{\mathbb{G}_s}^{\mathsf{ddh}}(t)$.*

Proof. We denote by $\mathrm{Adv}_{\mathcal{E}_s}^{\mathsf{ind\text{-}cpa}}(\mathcal{A})$ 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 $\mathrm{Exp}_{\mathcal{E}_s}^{\mathsf{ind\text{-}cpa\text{-}0}}(\mathcal{A})$, where $b = 0$:
$\mathcal{S}(\lambda)$:
- $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{Setup}(\lambda)$
- $(\mathsf{sk}_s, \mathsf{pk}_s) \leftarrow \mathsf{Keygen}_s(\mathsf{param})$
- $m_0, m_1, [\mathbf{a}]_s \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{pk}_s)$
- $C_s = (m_0 \cdot [\mathbf{a}]_s + r \cdot [\mathbf{p}_s]_s, [\mathbf{a}]_s) \leftarrow \mathsf{Encrypt}_s(\mathsf{pk}_s, m_0, [\mathbf{a}]_s)$
- $b' \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{pk}_s, C_s)$

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 $\mathbf{G}_1$: 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 $([\mathbf{p}]_s, [\mathbf{r}]_s)$: $\Pr_{\mathbf{G}_2}[E] - \Pr_{\mathbf{G}_1}[E] \leq \mathrm{Adv}_{\mathbb{G}_s}^{\mathsf{ddh}}(t)$.

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: $\Pr_{\mathbf{G}_4}[E] - \Pr_{\mathbf{G}_3}[E] \leq \mathrm{Adv}_{\mathbb{G}_s}^{\mathsf{ddh}}(t)$.

Game $\mathbf{G}_5$: In this game, the simulator can perfectly emulate the challenger in the experiment $\mathrm{Exp}_{\mathcal{E}_s}^{\mathsf{ind\text{-}cpa\text{-}1}}(\mathcal{A})$, 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, $\Pr_{\mathbf{G}_5}[E] = \Pr\left[\mathrm{Exp}_{\mathcal{E}_s}^{\mathsf{ind\text{-}cpa\text{-}1}}(\mathcal{A}) = 1\right]$, hence

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

which concludes the proof. □

Corollary 11. *$\mathcal{E}_T$ is* IND-CPA *under the* DDH *assumptions in $\mathbb{G}_1$ or $\mathbb{G}_2$. More precisely, for any adversary $\mathcal{A}$ running within time t,*

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

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 IND-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 $\mathrm{Exp}_{\mathcal{E}_T}^{\textsf{ind-cpa-0}}(\mathcal{A})$, where $b = 0$:
$\mathcal{S}(\lambda)$:
- $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{Setup}(\lambda)$
- $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{Keygen}_1(\mathsf{param})$,$(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{Keygen}_2(\mathsf{param})$
- $m_0, m_1, [\mathbf{a}]_1, [\mathbf{a}]_2 \leftarrow \mathcal{A}(\mathsf{param}, (\mathsf{pk}_1, \mathsf{pk}_2))$
- $C_T = \mathsf{Encrypt}_T((\mathsf{pk}_1, \mathsf{pk}_2), m_0, ([\mathbf{a}]_1, [\mathbf{a}]_2))$
- $\beta \leftarrow \mathcal{A}(\mathsf{param}, (\mathsf{pk}_1, \mathsf{pk}_2), C_T)$

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

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

Game $\mathbf{G}_1$: The simulator interacts with a challenger in $\mathrm{Exp}_{\mathcal{E}_1}^{\textsf{ind-cpa-0}}(\mathcal{A})$, 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 the ciphertext $C_T = \mathsf{Multiply}(C_1, \mathsf{Encrypt}_2(\mathsf{pk}_2, 1, [\mathbf{a}_2]_2))$, 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 $\mathrm{Exp}_{\mathcal{E}_1}^{\textsf{ind-cpa-1}}(\mathcal{A})$, 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, $\Pr_{\mathbf{G}_3}[E] = \Pr\left[\mathrm{Exp}_{\mathcal{E}_T}^{\textsf{ind-cpa-1}}(\mathcal{A}) = 1\right]$, hence

$$\Pr\left[\mathrm{Exp}_{\mathcal{E}_T}^{\textsf{ind-cpa-1}}(\mathcal{A}) = 1\right] - \Pr\left[\mathrm{Exp}_{\mathcal{E}_T}^{\textsf{ind-cpa-0}}(\mathcal{A}) = 1\right] \leq \mathrm{Adv}_{\mathcal{E}_T}^{\textsf{ind-cpa}}(t + t_m + t_e),$$

which concludes the proof, since it works exactly the same way for $\mathbb{G}_2$. □

A.8 Re-Encryption

We have three efficient encryption schemes able to compute homomorphic operations and supporting multiple users. Re-encryption, in order to transform a ciphertext for Alice into a ciphertext to Bob, might then be useful, as it will allow to target a specific end-user. With the Freeman's 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 into 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.

$\mathsf{REKeygen}_s(\mathsf{sk}_s^a, \mathsf{sk}_s^b)$**:** 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 $\mathbf{B}_s, \mathbf{B}'_s \in \mathrm{GL}_2(\mathbb{Z}_p)$ 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 $\mathsf{rk}_s^{a \to b} = \mathbf{R}_s = \mathbf{B}_s^{-1}\mathbf{B}'_s$ the secret re-encryption key. From the re-encryption keys $\mathsf{rk}_1^{a \to b} = \mathbf{R}_1 \leftarrow \mathsf{REKeygen}_1(\mathsf{sk}_1^a, \mathsf{sk}_1^b)$ and $\mathsf{rk}_2^{a \to b} = \mathbf{R}_2 \leftarrow \mathsf{REKeygen}_2(\mathsf{sk}_2^a, \mathsf{sk}_2^b)$, we will consider $\mathsf{rk}_T^{a \to b} = (\mathsf{rk}_1^{a \to b}, \mathsf{rk}_2^{a \to b})$, as the matrix $\mathbf{R}_T$ actually is $\mathbf{R}_1 \otimes \mathbf{R}_2$.
$\mathsf{Re\text{-}encrypt}_s(\mathsf{rk}_s^{a \to b}, C_s)$**:** 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 $([\mathbf{c}_{T,1}]_T \cdot (\mathsf{rk}_1^{a\to b} \otimes \mathsf{rk}_2^{a\to b}), [\mathbf{c}_{T,2}]_T \cdot (\mathsf{rk}_1^{a\to b} \otimes \mathsf{rk}_2^{a\to b}))$.

We stress that the basis $\mathbf{a}$ is modified with the re-encryption process, into $\mathbf{aR}_s$ 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. However, this requires this basis $\mathbf{a}$ to be part of the ciphertext as it cannot be a constant.

A.9 Correctness of Re-Encryption

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 $s \in \{1, 2\}$,

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

For $s = T$,

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

A.10 Distributed Decryption

When a third-party performs the decryption, it is important to be able to prove the correct decryption, which consists of zero-knowledge proofs, as described in the Appendix B. 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 or under the control of an adversary.

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 $I \subset \{1, \ldots, n\}$ 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.

For $s = T$ and with just the distribution of sk_1 and sk_2, it is also possible to perform a distributed decryption, using the relation $\mathsf{sk}_1 \otimes \mathsf{sk}_2 = (\mathsf{sk}_1 \otimes \mathbf{1}) \times (\mathbf{1} \otimes \mathsf{sk}_2)$. One can thus make a two round decryption, first in $\mathbb{G}_1$ and then in $\mathbb{G}_2$.

Remark 12. Because the operations to decrypt or re-encrypt are the same, one can make distributed re-encryption in the same vein: in our applications, computations will be performed on data encrypted under a *controller*'s key, where the *controller* is actually a pool of controllers with a distributed decryption key. The latter will be used to re-encrypt the result under the targer end-user's key.

However, in this scheme, the secret key must be a projection matrix, which is not easy to generate at random: for this key generation algorithm, a trusted dealer is required, which is not ideal when nobody is trusted. This is the goal of the rest of the paper, to show that we can optimize this generic construction, and distribute everything without any trusted dealer.

B 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 13. 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 $\mathbf{B}_s \xleftarrow{\$} \mathrm{GL}_2(\mathbb{Z}_p)$. 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 $\mathbf{P}'_s = \mathbf{B}'^{-1}_s\mathbf{U}\mathbf{B}'_s$. 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.

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 $\mathsf{Randomize}_s$. 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 $\mathsf{Decrypt}_s$. 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 $\mathsf{rk}_s^{a\to b} = \mathbf{R}_s \in \mathrm{GL}_2(\mathbb{Z}_p)$ 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 $\mathsf{rk}_T^{a\to b} = (\mathbf{R}_1, \mathbf{R}_2) \in \mathrm{GL}_2(\mathbb{Z}_p)^2$ 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 14. *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 10:

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,

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

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 $([\mathbf{a}]_s, [\mathbf{b}]_s)$: $\Pr_{\mathbf{G}_3}[E] - \Pr_{\mathbf{G}_2}[E] \leq \mathrm{Adv}_{\mathbb{G}_s}^{\mathsf{ddh}}(t)$. 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 $\mathrm{Exp}_{\mathcal{E}_s}^{\mathsf{ind\text{-}cpa\text{-}1}}(\mathcal{A})$.

Hence, we have:

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

□

Corollary 15. *$\mathcal{E}_T$ 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. □