Secure Distributed Computation on Private Inputs

Geoffroy Couteau, Thomas Peters, and David Pointcheval

ENS, CNRS & INRIA, PSL Research University, Paris, France

Abstract. The recent notion of encryption switching protocol (ESP) allows two players to obliviously switch between two encryption schemes. Instantiated from multiplicatively homomorphic encryption and additively homomorphic encryption, ESPs provide a generic solution to two-party computation and lead to particularly efficient protocols for arithmetic circuits in terms of interaction and communication.
In this paper, we further investigate their applications and show how ESPs can be used as an alternative to fully-homomorphic encryption (FHE) to outsource computation on sensitive data to cloud providers. Our interactive solution relies on two non-colluding servers which obliviously perform the operations on encrypted data, and eventually send back the outcome in an encrypted form to the appropriate players. Our solution makes use of a nice combination of the Paillier encryption scheme and the Damgard-Jurik variant with multiple trapdoors, which notably allows cross-user evaluations on encrypted data.

Keywords. encryption switching protocols, delegation of computations.

1 Introduction

Secure multiparty computation (MPC) targets the following problem: several players, modeled as probabilistic polynomial time Turing machines, wish to jointly compute a public function f of their respective, private inputs. The players want to guarantee the privacy of their inputs: at the end of the protocol, no player should be able to deduce from its view during the execution anything that he could not have deduced from the output and its own input. General solutions for MPC have been proposed, starting with the work of Yao [Yao86] and followed by [GMW87b, GMW87a].

Delegation of Computations. A particular case of secure MPC is the problem of delegation of computations where a set of clients, with a limited computational power, would like to perform some expensive computation on their data. To do so, they can communicate with a powerful server, but they do not fully trust the server. Typical security requirements for this particular task are both the *privacy*, which says that the server will perform computations on the inputs of the clients without gaining any knowledge about this input, and the *verifiability*, which says that the clients want to be sure that the correct functions have been evaluated. In this paper, we propose a solution which guarantees both requirements in a specific setting, namely the non-collusion of the two servers.

Encryption Switching Protocols. An encryption switching protocol (ESP) is a protocol which allows two players to obliviously switch from a ciphertext under one encryption scheme to a ciphertext encrypting the same plaintext under the other encryption scheme. We recall the definition of ESP as introduced in [GC15]: let $(\mathcal{E}, \mathcal{D})$ and $(\overline{\mathcal{E}}, \overline{\mathcal{D}})$ be two encryption schemes, with a common pair of encryption and decryption keys $(\mathsf{pk}, \mathsf{sk})$ —in any case, they can always be the concatenations of the keys of the two schemes—. An ESP between two players, each player holding a share of sk, on input an encryption C of some plaintext m under $\mathcal{E}$, outputs an encryption $\bar{C}$ of the same plaintext m under $\overline{\mathcal{E}}$. Informally, an ESP is *secure* if it satisfies the following requirements:

- **Correctness**: if both players behave honestly, $\overline{\mathcal{D}}(\bar{C}, \mathsf{sk}) = \mathcal{D}(C, \mathsf{sk})$;
- **Soundness**: there is a negligible probability for a malicious player to force an output $\bar{C}$ such that $\overline{\mathcal{D}}(\bar{C}, \mathsf{sk}) \neq \mathcal{D}(C, \mathsf{sk})$ without being detected;
- **Zero-Knowledge**: no information on either m or sk leaks during the protocol execution.

It should be noted that, as the players hold *shares* of the secret key sk, the decryption and encryption switching operations are performed as two-party protocols between the players. They essentially both involve $(2, 2)$-threshold decryption.

Two-Party Computation from ESP. In [GC15], the authors established that any function on a ring $(\mathcal{R}, +, \times)$ can be evaluated on encrypted data by a two-party protocol relying on two encryption schemes, an additively homomorphic scheme $(\mathcal{E}_\oplus, \mathcal{D}_\oplus)$ and a multiplicatively homomorphic scheme $(\mathcal{E}_\otimes, \mathcal{D}_\otimes)$, operating on the same plaintext space $\mathcal{R}$, with an ESP to obliviously switch between those two schemes. We recall the intuition underlying this result by considering two players, Alice and Bob, who wish to evaluate a N-variate polynomial $P(X_1, \cdots, X_N) = \sum_{j=1}^{M} \prod X_i^{d_{ij}}$ in $\mathcal{R}[X_1, \ldots, X_N]$ on a vector of inputs $\boldsymbol{x}$ shared between the two parties. Let $\boldsymbol{x_A} = (x_A^i)_{i \leq N}$ (resp. $\boldsymbol{x_B} = (x_B^i)_{i \leq N}$) be Alice's share (resp. Bob's share), such that $\boldsymbol{x} = \boldsymbol{x_A} + \boldsymbol{x_B}$, in the ring:

1. Both players start by encrypting their inputs under $\mathcal{E}_\oplus$, the additively homomorphic encryption scheme. If we denote $\boxplus$ the homomorphic operator on the ciphertexts for the $+$ operator on the plaintexts, both players can then compute $\mathcal{E}_\oplus(\boldsymbol{x}) = \mathcal{E}_\oplus(\boldsymbol{x_A}) \boxplus \mathcal{E}_\oplus(\boldsymbol{x_B}) = (\mathcal{E}_\oplus(x_i))_{i \leq N}$;
2. Both players then perform N ESPs in parallel to get a vector $\mathcal{E}_\otimes(\boldsymbol{x})$ of multiplicatively homomorphic ciphertexts. Let $\boxtimes$ and $\triangle$ be the homomorphic operators for multiplication and exponentiation by a constant on the plaintexts, then both players can compute $(\mathcal{E}_\otimes(y_j))_{j \leq M} = (\mathcal{E}_\otimes(\prod_i x_i^{d_{ij}}))_{j \leq M} = (\boxtimes_{i \leq N} \mathcal{E}_\otimes(x_i) \triangle d_{ij})_{j \leq M}$;
3. Alice and Bob eventually perform M ESPs in parallel to get additively homomorphic encryptions of all the monomials y_j. They compute, from the previous result, an encryption of the sum of all the monomials, which is the target output $P(x_1, \cdots, x_N)$, that they decrypt in a distributed way.

The overall protocol involves two steps of switches (all the other operations are performed locally by each player), with a communication of $O(N + M)$ ciphertexts: this method avoids the dependency in the degree of the polynomial which is inherent to most two-party protocols, and is round-efficient (i.e., independent of the d_{ij}'s, while known protocols for this task typically involve $O(\log(\max_{i,j}(d_{ij})))$ rounds of interaction).

Our Contribution. In this paper, we design a novel application of ESP to delegation of computations in the cloud. Specifically, we consider two non-colluding servers and design a delegation model with the following features:

- Any number of users can generate their own encryption and decryption keys;
- Any player can store its data online, encrypted under its own key;
- Any subset of users can ask for the evaluation of an arbitrary function f over their joint data;
- The communication and interactions between the servers are independent of the depth of the arithmetic circuit to be evaluated, to avoid latency issues;
- If the servers do not collude, then the protocol guarantees the privacy of the data and the correctness of the computation.

For efficiency reasons, we want the computation of the servers to be small, compared to solutions based on fully-homomorphic encryption [Gen09]. But still, our model expects a single round with the cloud providers. It means that the users' encrypted data are sent and stored before any evaluation. At the end of the evaluation, using the stand-alone two-party computation [GC15] from ESP, all the results are sent to the appropriate users and the protocol is over.

2 Delegation to Non-Colluding Servers

2.1 Delegating Computations to the Cloud

We consider the following scenario: many individuals, with very limited computational resources, would like to securely outsource their data and to be able to delegate computations on these data to the more powerful cloud providers. As the data are possibly sensitive, it should be guaranteed that the privacy of the inputs of each user will be preserved, preferably without restricting the kind of computation that can be outsourced.

A natural solution to this problem is fully-homomorphic encryption (FHE): each user generates his own secret key for an FHE scheme; to outsource its data, the user encrypts his input and sends it to the cloud. When he wants to delegate computations, the user simply sends the target function to the

server. Thanks to the homomorphic properties of the scheme, the latter can evaluate any function on the encrypted input and send back the encrypted result to the user, which decrypts it to get his output. Even if secret-key fully-homomorphic encryption is enough in such a context, current schemes remain prohibitively expensive, making this solution impractical for complex functions, which are typically the kind of functions for which a user would like to delegate computations. Moreover, evaluating functions on joint data of several clients, encrypted under different keys, requires the use of a FHE with stronger properties, namely multi-key FHE [LTV12].

An alternative solution to this problem can be obtained by considering two non-colluding servers (or more) instead of a single one. Indeed, it would be natural to see two cloud providers collaborate in order to offer such a service to their customers, but they would arguably not share their own private data to the other provider, as they do not trust each other. In this setting, the two providers could run a two-party protocol on the encrypted data to get an encrypted output, which would be sent back to the user. However, this approach raises two concerns:

1. Multiparty computation protocols to evaluate function on encrypted data typically require the players (here, the cloud providers) to hold shares of the secret key of the encryption scheme, while the scenario under consideration involves many users who wish to outsource data encrypted with their *own* key and to be able to decrypt the result using their own private key;
2. Two-party computation protocols require many interactions (typically, the players exchange messages for each product gate of the circuit to be evaluated), and the scenario we consider involves possibly distant, on-line servers. Therefore, a large number of interactions is highly undesirable because of the latency of the network.

2.2 The Model

In this paper, we propose a solution to address the constraints of the above scenario. In a nutshell, our solution works as follows: we rely on the DJ encryption scheme, introduced by Damgard and Jurik in [DJ03], which improves upon a previous similar scheme of [CS02]. DJ is an encryption scheme with a particular *double trapdoor* mechanism which ensures that any ciphertext can be decrypted in two ways, either with a *local* secret key specific to a player, or with the *global* master secrey key. A DJ encryption scheme is set up, and the master public key mpk (which is an RSA modulus) is made publicly available. Each player P_i can now generate its own secret key sk_i, and the corresponding public key pk_i. Any message encrypted under the public key pk_i of a player P_i can therefore be decrypted in two ways: either using sk_i, the secret key associated to pk_i generated by P_i, or by using the master secret secret key msk, which can decrypt ciphertext regardless of the particular key pk_i which was used to encrypt it.

The master secret key is divided into two shares, msk_1 and msk_2, using a secret sharing scheme; hence, each msk_j for $j \in \{1, 2\}$ does not reveal (statistically) any information on msk. Each of the two servers receives one share. With these shares, the two servers can now jointly compute functions over data encrypted under *any* particular key pk_i of a player P_i.

Let S_1 and S_2 be the two non-colluding servers to which the computation will be delegated. All the players encrypt their data under their own key pk_i and outsource the ciphertexts. When a subset S of players $(P_i)_{i \in S}$ asks for the evaluation of some function f on some vector $\boldsymbol{C}$ of data (encrypted under different keys if S contains at least two users), S_1 and S_2 first convert $\boldsymbol{C}$ into a vector $\boldsymbol{C}_+$ of Paillier ciphertexts using a tailored protocol which takes as input their shares of the master secret key msk. Second, the target function f is divided into *layers*, each layer containing either linear operations, or products and exponentiations. To evaluate a layer of linear operations on the encrypted data, S_1 and S_2 rely on the homomorphic properties of the Paillier encryption scheme, and can thus independently make the evaluation in a deterministic way, to get the same results. To evaluate a layer of products and exponentiations, S_1 and S_2 first interactively convert $\boldsymbol{C}_+$ into a vector $\boldsymbol{C}_\times$ of *multiplicatively* homomorphic ciphertexts, using an encryption switching protocol; once this is done, S_1 and S_2 can independently evaluate the operations on $\boldsymbol{C}_\times$ using the homomorphic properties of the scheme. They can then interactively switch the result back to additively homomorphic ciphertexts to evaluate linear operations, and so on, until the last layer of operations is performed. The result is then switched back

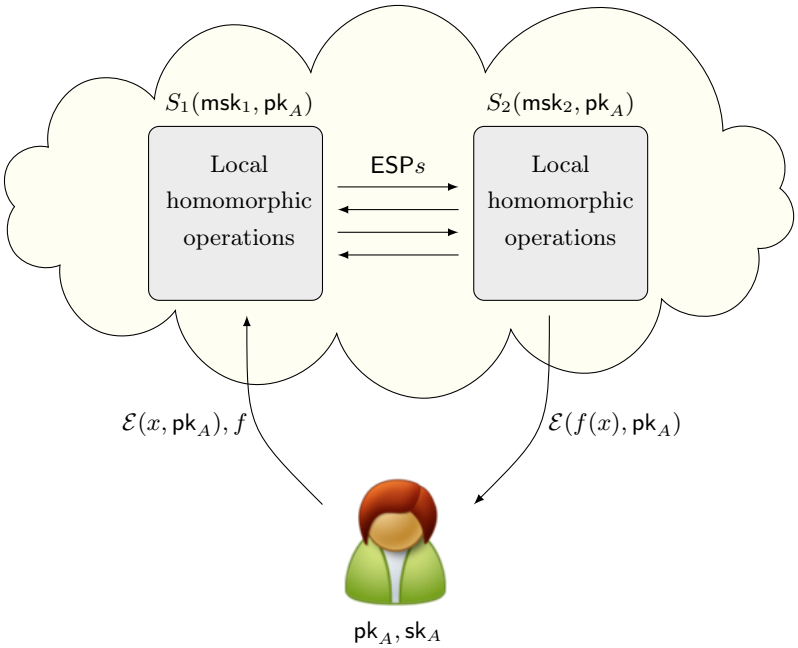

Fig. 1: Delegation of computation to two non-colluding servers

to a vector of DJ ciphertexts, encrypted under the key pk_i of each P_i for $i \in S$, and sent to those players. The model is represented Figure 1, in the case of a unique player in S.

Since computations are performed on ciphertexts, the privacy of the data is guaranteed. Since both servers do the same homomorphic operations, and the encryption switching protocols are secure against malicious players, if they both output the same result, it is necessarily correct, hence the verifiability, under the non-colluding servers assumption.

3 Formal Construction

We write $a \leftarrow b$ to affect the value of b to the variable a, and $a \xleftarrow{\$} D$ to affect a value sampled uniformly at random from the distribution D to the variable a. To avoid confusions between modular integers and integers, we use brackets to denote integers: $[a \bmod t]$ denotes the value of $a \bmod t$, between 0 and $t-1$, seen as an integer.

Global Setup. In all the constructions, we assume that the following global setup has been performed: a trusted dealer generates two primes p and q such that $p' = (p-1)/2$ and $q' = (q-1)/2$ are also prime, and sets $n \leftarrow pq$. He sets $\mathsf{mpk} \leftarrow n$ and $\mathsf{sk} \leftarrow (p, q)$. Let $\lambda \leftarrow (p-1)(q-1)/2$ be the maximal order of an element in the multiplicative subgroup $\mathbb{Z}_n^*$.

Jacobi Symbol. The Jacobi symbol is an extension of the Legendre symbol to groups with odd order. It maps invertible elements of the group to $\{-1, +1\}$. The Jacobi symbol of an element of $\mathbb{Z}_n^*$ can be efficiently computed, even when the factorization of the modulus is unknown.

3.1 Assumptions

We review the various assumptions on which the security of the primitives rely.

The Decisional Diffie-Hellman Assumption (DDH). Let $\mathbb{G}$ be some group of order t with generator g. The DDH assumption over $\mathbb{G}$ states that it is computationally infeasible to distinguish the following distributions:

$$\mathcal{D}_0 = \{(A, B, C) \mid (a, b, c) \xleftarrow{\$} \mathbb{Z}_t^3, (A, B, C) \leftarrow (g^a, g^b, g^c)\}$$
$$\mathcal{D}_1 = \{(A, B, C) \mid (a, b) \xleftarrow{\$} \mathbb{Z}_t^2, (A, B, C) \leftarrow (g^a, g^b, g^{ab})\}$$

For any integer t, let us write $\mathsf{QR}(t)$ the group of squares over $\mathbb{Z}_t^*$. The DDH assumption is conjectured to hold in the groups $\mathsf{QR}(n)$ and $\mathsf{QR}(n^2)$, for RSA moduli n, as defined above in the global setup.

The Quadratic Residuosity Assumption (QR). Let $\mathbb{J}_n$ be the group of elements of $\mathbb{Z}_n^*$ with Jacobi symbol $+1$; it holds that $\mathbb{J}_n = \{x \in \mathbb{Z}_n^* \mid (u,b) \leftarrow \mathbb{Z}_n^* \times \{0,1\}, x \leftarrow (-1)^b u^2\}$. The QR assumption modulo n states that it is computationally infeasible, given some element $x \xleftarrow{\$} \mathbb{J}_n$, to have a non-negligible advantage in guessing whether $x \in \mathsf{QR}(n)$ or $x \in \mathbb{J}_n \setminus \mathsf{QR}(n)$. Note that given the factorization of n, it is easy to break the QR assumption modulo n. Since an element $y \in \mathbb{Z}_{n^2}^*$ can be written as $y = x^n \cdot (1+rn) \bmod n^2$, with $x \in \mathbb{Z}_n^*$ and $r \in \mathbb{Z}_n$, and $1+rn$ is always a square in $\mathbb{Z}_{n^2}^*$, $y \in \mathsf{QR}(n^2)$ if and only if $x \in \mathsf{QR}(n)$, or equivalently $x^n = y \bmod n \in \mathsf{QR}(n)$. Hence, the QR assumptions are equivalent modulo n and modulo n^2, and so the unique notation QR.

The Decisional Composite Residuosity Assumption (DCR). Let $D_n = \{x \in \mathbb{Z}_{n^2}^* \mid \exists y \in \mathbb{Z}_n^*, x = y^n \bmod n^2\}$. The DCR assumption states that it is computationally infeasible, given an element x sampled at random from either $\mathbb{Z}_{n^2}^*$ or D_n, to guess whether $x \in D_n$ or not with non-negligible advantage over the random guess.

3.2 Primitives

We recall the descriptions of the primitives which are used in the construction. For convenience, we will slightly abuse the usual notations for encryption, and write $\mathsf{Enc}(m)$ instead of $\mathsf{Enc}(m;r)$ with a random coin r.

Paillier Encryption Scheme. The Paillier encryption scheme was introduced in [Pai99]. The global setup is run; let $\mathsf{msk} := d \leftarrow [\lambda^{-1} \bmod n] \times \lambda \bmod n\lambda$; note that the knowledge of sk implies the knowledge of d.

$\mathsf{Enc}(\mathsf{pk}, m)$: On input $m \in \mathbb{Z}_n$, pick $r \xleftarrow{\$} \mathbb{Z}_{n^2}$ and output $c = \mathcal{E}_{\oplus}(\mathsf{pk}, m; r) = (1+nm) \cdot r^n \bmod n^2$;
$\mathsf{Dec}(\mathsf{sk}, c)$: on input c, output $m = ([c^d \bmod n^2] - 1)/n$

This scheme is $\mathsf{IND\text{-}CPA}$ under the DCR assumption over $\mathbb{Z}_{n^2}^*$, and additively homomorphic in $\mathbb{Z}_n$.

Damgard-Jurik Encryption Scheme. The Damgard-Jurik scheme (DJ) was introduced in [DJ03]. It builds upon the Paillier encryption scheme and enhances it with a double trapdoor mechanism: for a fixed ring $\mathbb{Z}_n$, many pairs of public and secret keys can be generated. This property was explicitly identified in [BCP03], in which a similar scheme with comparable properties under weaker assumptions was constructed. A given ciphertext encrypted with a public key pk_i can be decrypted in two ways, either with the associated secret key sk_i in an ElGamal-like fashion, or using the master secret key, i.e., the Paillier's decryption key. We denote $\mathcal{E}_{\oplus}^i$ the DJ scheme for user P_i. Let G be a generator of $\mathbb{J}_n$ added to mpk.

$\mathsf{Key}(\mathsf{mpk}, i)$: Pick $a_i \xleftarrow{\$} \mathbb{Z}_{n/2}$ and set $H_i \leftarrow G^{a_i} \bmod n$. Let $\mathsf{pk}_i \leftarrow H_i$ be the public key of the player P_i. The secret key is $\mathsf{sk}_i \leftarrow a_i$;
$\mathsf{Enc}(\mathsf{mpk}, \mathsf{pk}_i, m)$: On input $m \in \mathbb{Z}_n$, pick $r \xleftarrow{\$} \mathbb{Z}_{n/2}$ and output $c = \mathcal{E}_{\oplus}^i(m; r) = (G^r \bmod n, (1+mn) \cdot [H_i^r \bmod n]^n \bmod n^2)$;
$\mathsf{Dec}(\mathsf{sk}_i, c)$: Parse c as (A, B) and output $m = ([B^2 \cdot [A^{2a_i} \bmod n]^{-n} \bmod n^2] - 1)/n \cdot [2^{-1} \bmod n] \bmod n$.

Alternatively, one can decrypt a ciphertext c with the master secret key $\mathsf{msk} = d$ simply by dropping the first part A of the ciphertext, and by decrypting B using the Paillier decryption procedure, since the mask $[H_i^r \bmod n]^n \bmod n^2$ is an n-th power in $\mathbb{Z}_{n^2}^*$.

This scheme is a slight variant of the *proof-friendly* encryption introduced in [DJ03]. Indeed, in their first scheme, G and H_i are drawn from $\mathsf{QR}(n)$ instead of $\mathbb{J}_n$. The proof-friendly variant lowers the complexity of zero-knowledge proofs of validity (or knowledge), at the very moderate cost of adding the QR assumption to the security: by squaring A and B in the decryption process, we ensure to work

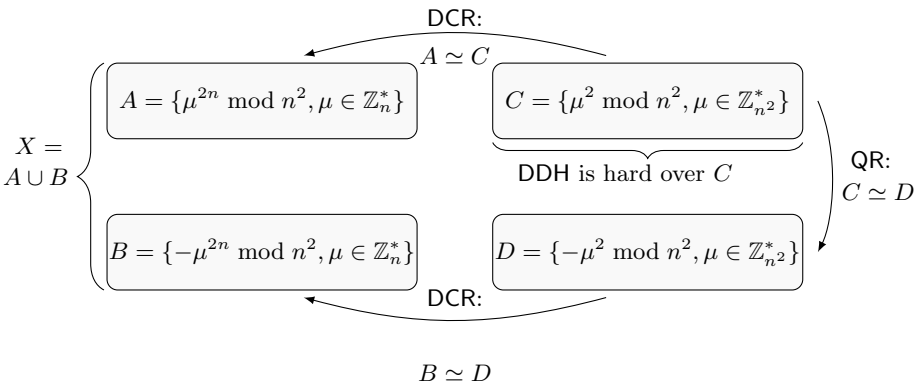

Fig. 2: Hardness of DDH over X

in a group without any small subgroup. This allows to decrypt $2m$ instead of m, but the factor 2 can be easily removed over $\mathbb{Z}_n$ (as inverses are easy to compute). In our construction, proofs of knowledge will be required by the servers to prevent malicious clients from trying to outsource data that they do not own. Hence, we favor this variant of the scheme in our application to reduce the work of the client in the model. Note that our construction makes use of another scheme, the $\mathbb{Z}_n^*$-EG encryption scheme which will be described afterward, whose security does already rely on the QR assumption. Thereby, adding the QR assumption to the DJ scheme does not affect the overall security of our construction.

On the IND-CPA security of the DJ *scheme.* The security of the scheme is proven in [DJ03]. In a nutshell, it relies on the DDH assumption over $\mathsf{QR}(n^2)$, the QR assumption, and the DCR assumption. Indeed, $[H_i^r \bmod n]^n \bmod n^2$ is in $X = \{\pm\mu^{2n}, \mu \in \mathbb{Z}_n^*\}$, hence the DDH assumption over X ensures that this value masks the plaintext. Figure 2 shows why (informally) the DDH assumption holds over X if it holds over $\mathsf{QR}(n^2)$ and if the DCR and QR assumptions hold.

$\mathbb{Z}_n^*$-EG Encryption Scheme. The $\mathbb{Z}_n^*$-EG scheme, denoted $\mathcal{E}_\otimes$ in this paper, is a multiplicatively homomorphic extension of the ElGamal encryption scheme [ElG85] over $\mathbb{Z}_n^*$. It makes a black-box use of an ElGamal scheme over the group $\mathbb{J}_n$ of elements with Jacobi symbol $+1$, denoted $\mathbb{J}_n$-EG, with encryption and decryption algorithms $\mathbb{J}_n$-EG.Enc and $\mathbb{J}_n$-EG.Dec.

$\mathsf{Setup}(p, q)$: Pick $g_0 \xleftarrow{\$} \mathbb{Z}_n^*$, $s \xleftarrow{\$} \mathbb{Z}_\lambda$, an even $t_p \xleftarrow{\$} \mathbb{Z}_\lambda$, and an odd $t_q \xleftarrow{\$} \mathbb{Z}_\lambda$; set $g \leftarrow -g_0^2$ (a generator of $\mathbb{J}_n$, of order λ), $v \leftarrow [p^{-1} \bmod q] \cdot p \bmod n$ ($v = 0 \bmod p$ and $v = 1 \bmod q$) and $\chi \leftarrow (1-v) \cdot g^{t_p} + v \cdot g^{t_q} \bmod n$, and $g_1 \leftarrow g^s \bmod n$ (for the $\mathbb{J}_n$-EG encryption);

$\mathsf{Enc}(\mathsf{pk}, m)$: On input $m \in \mathbb{Z}_n^*$, compute $(m_1, m_2) \leftarrow (g^a, \chi^{-a}m) \in {\mathbb{J}_n}^2$ for $a \xleftarrow{\$} \mathbb{Z}_{n/2}$, so that $J_n(m) = (-1)^a$. Then, choose $r \xleftarrow{\$} \mathbb{Z}_{n/2}$ and compute $C \leftarrow \mathbb{J}_n\text{-EG.Enc}(m_2; r) = (c_0 = g^r,\ c_1 = m_2 g_1^r)$. Return the ciphertext $c \leftarrow \mathcal{E}_\otimes(m; r) = (C = (c_0, c_1), m_1)$;

$\mathsf{Dec}(\mathsf{sk}, c)$: Parse $c = (C = (c_0, c_1), m_1)$ and check whether $J_n(c_1) = 1$. If not, return $\perp$, otherwise compute $m_2 \leftarrow \mathbb{J}_n\text{-EG.Dec}(C) = c_1/c_0^s$ in $\mathbb{Z}_n^*$ and then $m_0 \leftarrow (1-v) \cdot m_1^{t_p} + v \cdot m_1^{t_q} \bmod n$. Return $m \leftarrow m_0 m_2 \bmod n$.

This scheme is IND-CPA under the QR assumption and the DDH assumption over $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$.

$\mathbb{Z}_n$-EG Encryption Scheme. The $\mathbb{Z}_n$-EG scheme, denoted $\mathcal{E}_\otimes^0$, is an extension of the $\mathbb{Z}_n^*$-EG scheme to $\mathbb{Z}_n^* \cup \{0\}$. The intuition of the extension is the following: a plaintext $m \in \mathbb{Z}_n^* \cup \{0\}$ is encoded as $(m + b, R^b)$, where b is a bit equal to 1 if and only if $m = 0$, and R is a random square (or in $\mathbb{J}_n$). It is easy to check that this encoding is multiplicatively homomorphic.

$\mathsf{Enc}(\mathsf{pk}, m)$: On input $m \in \mathbb{Z}_n^*$, set $b = 1$ if $m = 0$, $b = 0$ else. Picks $(R, R') \xleftarrow{\$} \mathbb{J}_n^2$. Return the ciphertext $\mathcal{E}_\otimes^0(m) \leftarrow (\mathcal{E}_\otimes(m + b), \mathbb{J}_n\text{-EG.Enc}(R^b), \mathbb{J}_n\text{-EG.Enc}(R'^b))$.

$\mathsf{Dec}(\mathsf{sk}, c)$: Decrypt the second ciphertext. If it contains any value $R \neq 1$, output 0. If it contains 1, decrypt the first ciphertext and output the result.

The scheme is IND-CPA secure: its IND-CPA security reduces to the IND-CPA security of $\mathbb{J}_n$-EG and $\mathbb{Z}_n^*$-EG. Note that when the decryption of the second part of the ciphertext returns $R \neq 1$, decrypting the first part might compromise the security. This is the reason which the latter should only be decrypted if the former is different from 1. Note also that the third component of the ciphertext is not used in the decryption process; in fact, the scheme would be still secure without this last component. However, this third component is necessary for the encryption switching protocol between this scheme and the Paillier encryption scheme. Since we will do two-party decryption, this will be guaranteed under the non-collusion of the two players.

It is important to note that finding an element outside of the plaintext space $\mathbb{Z}_n^* \cup \{0\}$ of the $\mathbb{Z}_n$-EG scheme is equivalent to factoring; hence, when the secret key is secretly shared between the two players and no single player knows the factorization, with overwhelming probability, no elements outside $\mathbb{Z}_n^* \cup \{0\}$ will ever be exchanged, making the plaintext space of the $\mathbb{Z}_n$-EG scheme equivalent *in practice* to the plaintext space $\mathbb{Z}_n$ of the Paillier scheme: this result implies that ESPs will not compromise the privacy of the plaintexts.

Threshold Schemes. The Paillier scheme and the $\mathbb{Z}_n$-EG scheme admit two-party threshold versions (see [HMRT12] and [GC15]), in which the secret key is shared between the players and the decryption is performed as a two-party protocol (in fact, the $\mathbb{Z}_n$-EG scheme is secure only in the threshold setting, as the first part of a ciphertext must not be decrypted if the second part does not encrypt 1, which can only be guaranteed when the key is shared). As a consequence, the DJ scheme does also admit a threshold decryption procedure for the alternative decryption procedure, using the master secret key: the players drop the first component of the ciphertext and perform a two-party Paillier decryption procedure on the second component.

Overview of ESP between Paillier and $\mathbb{Z}_n$-EG (Informal). We outline the intuition underlying the ESP protocol: to switch from a ciphertext C from an encryption scheme E_1 to the other encryption scheme E_2, Alice picks a random mask, uses it to mask C and decrypts the resulting ciphertext with Bob (Bob gets the result). Bob encrypts the outcome under E_2; the mask is then homomorphically removed in the ciphertext space of the second scheme.

3.3 Protocol

A trusted dealer performs the global setup described Section 3.2 and generates the secret key $d \in \mathbb{Z}_{n\lambda}$ of the Paillier scheme, the secret key $(v, t_p, t_q, s) \in \mathbb{Z}_n \times \mathbb{Z}_\lambda^3$ of the $\mathbb{Z}_n$-EG scheme; he sets $\mathsf{msk} \leftarrow d$ for the DJ scheme. The dealer secretly shares these keys into two tuples of shares, such that each tuple does individually not reveal anything about the secret key. The dealer sends the first share to the first server S_1 and the second share to the second server S_2. Then, he broadcasts the RSA modulus n.

Outsourcing the Data. To outsource his data, a player P_i performs the Setup algorithm of the DJ scheme on input (n, i) to get $(\mathsf{pk}_i, \mathsf{sk}_i)$. Let $\boldsymbol{x} = (x_1, \cdots, x_N)$ be his data; then

- P_i computes $\boldsymbol{C} \leftarrow \mathcal{E}_\oplus^i(\boldsymbol{x}) = \mathcal{E}_\oplus^i(x_1), \cdots, \mathcal{E}_\oplus^i(x_N)$ and sends it to S_1.
- P_i proves, in zero-knowledge, that he knows plaintexts such that all the ciphertext are encryptions with pk_i of those plaintexts.

On the Zero-Knowledge Proof of Knowledge. The last step ensures that no player will be able to "steal" data from other players by sending a (potentially re-randomized) ciphertext of an other player to the servers as being its own data. Note that the proofs are classical proofs of knowledge of a representation, *à la* Schnorr [Sch90].

One can consider two variants: the player can perform an interactive proof of knowledge with each server, and the servers will let him outsource the data if the proofs succeed. Alternatively, the player can simply append a non-interactive proof to the data he sends, so that the data can be sent to a single server and checked by the two servers when they are asked to perform computation over them. This latter method however requires the random oracle model.

Note that the communication of the zero-knowledge proof can be made independent of the number of inputs by proving the knowledge of the plaintext of the ciphertext $\boxplus_j \lambda_j \mathcal{E}^i_\oplus(x_j)$, where the λ_j's can be derived with either a pseudo-random generator from a seed sent by the verifier or a random oracle on the statement. If the proof succeeds then, with overwhelming probability, all the ciphertexts are valid encryptions with pk_i and P_i knows all the corresponding plaintexts. Note however that this method implies a loss linear in the number of ciphertexts in the security reduction.

Evaluating a Function on Encrypted Data. A subset S of players $(P_i)_{i\in S}$ sends a target function f to the servers S_1 and S_2. For convenience, we assume that f is written in a *layered* form, i.e., as a sequence of layers, each layer containing either linear combinations or monomials computations. First, the servers convert the DJ ciphertexts into Paillier ciphertexts (simply by dropping the first component of each ciphertext). Once this is done, the function is evaluated each layer at a time, by first switching to the encryption scheme with the appropriate homomorphism (using a secure ESP between the Paillier scheme and the $\mathbb{Z}_n$-EG scheme) and evaluating the layer locally and homomorphically. It follows immediately from the Theorem 8 of [GC15] (which states that two-party computation from ESP is secure) that if at least one of the servers is honest, then:

- The output is a (vector of) Paillier ciphertext(s) which encrypts $f(\boldsymbol{x})$;
- The view of both servers can be simulated without $\boldsymbol{x}$ or the shares of the secret key (hence the privacy of $\boldsymbol{x}$ is guaranteed).

The number of rounds of the protocol is proportional to the number of layers of f (typically, this number is 2 for a multivariate polynomial represented by its canonical form, and will be independent of the degree of f in most practical cases).

Sending the Result Back to P_i for each $i \in S$. For each Paillier ciphertext C obtained as output of the evaluation of f on the encrypted data of the players $(P_i)_{i\in S}$, S_1 and S_2 run the following protocol for each $i \in S$ to convert it into a DJ ciphertext under the key of P_i:

1. S_1 picks $r \xleftarrow{\$} \mathbb{Z}_n$ and sends $C_1 \leftarrow C \boxplus \mathcal{E}_\oplus(r)$ and $C_2 \leftarrow \mathcal{E}^i_\oplus(-r)$ to S_2;
2. S_1 and S_2 jointly decrypt C_1; S_2 gets the result t and sends back $C_i \leftarrow \mathcal{E}^i_\oplus(t) \boxplus C_2$;
3. C_i is sent to P_i, which decrypts it using sk_i.

Note that the protocol has been described in the honest-but-curious setting: malicious servers might deviate from the specifications of the protocol. However, the security can be easily enhanced into full security: the security against malicious adversaries can be ensured by asking S_1 to commit to r and prove the consistency of C_1 and C_2 with the commitment using zero-knowledge proofs. S_2 can also prove, in zero-knowledge, that C' was constructed consistently from C_2 and the plaintext of C_1. Then, both servers send back the result to P_i; if they are not equal, P_i ignores the output. In this setting, it is sufficient that one of the two servers is being honest to ensure that the privacy of P_i's input is guaranteed, and that the output of the protocol is indeed $f(\boldsymbol{x})$.

Overall, this model ensures the privacy of the data in the stand-alone setting, for a single run of the function evaluation protocol. It would be an interesting future direction of work to extend that to multiple adaptive evaluations on dynamically outsourced data.

Acknowledgments

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

References

BCP03. E. Bresson, D. Catalano, and D. Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In *ASIACRYPT 2003*, *LNCS* 2894, pages 37–54. Springer, November / December 2003.

CS02. R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT 2002*, *LNCS* 2332, pages 45–64. Springer, April / May 2002.

DJ03. I. Damgård and M. Jurik. A length-flexible threshold cryptosystem with applications. In *ACISP 03*, *LNCS* 2727, pages 350–364. Springer, July 2003.

ElG85. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

GC15. D. P. Geoffroy Couteau, Thomas Peters. Encryption switching protocols. Cryptology ePrint Archive, Report 2015/990, 2015. `http://eprint.iacr.org/`.

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

GMW87a. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

GMW87b. O. Goldreich, S. Micali, and A. Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *CRYPTO'86*, *LNCS* 263, pages 171–185. Springer, August 1987.

HMRT12. C. Hazay, G. L. Mikkelsen, T. Rabin, and T. Toft. Efficient RSA key generation and threshold Paillier in the two-party setting. In *CT-RSA 2012*, *LNCS* 7178, pages 313–331. Springer, February / March 2012.

LTV12. A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012.

Pai99. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, *LNCS* 1592, pages 223–238. Springer, May 1999.

Sch90. C.-P. Schnorr. Efficient identification and signatures for smart cards (abstract) (rump session). In *EUROCRYPT'89*, *LNCS* 434, pages 688–689. Springer, April 1990.

Yao86. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.